Information-Theoretic Analysis of Molecular (Co)Evolution Using Graphics Processing Units

Michael Waechter
Interactive Graphics Systems Group
Dept. of Computer Science
TU Darmstadt, Germany

Kathrin Jaeger
Bioinformatics Group
Department of Biology
TU Darmstadt, Germany

Stephanie Weissgraeber
Bioinformatics Group
Department of Biology
TU Darmstadt, Germany

Sven Widmer
Interactive Graphics Systems Group
Dept. of Computer Science
TU Darmstadt, Germany

Michael Goesele
Interactive Graphics Systems Group
Dept. of Computer Science
TU Darmstadt, Germany

Kay Hamacher*
Bioinformatics Group
Depts. of Biology, Physics & Computer Science
TU Darmstadt, Germany
hamacher@bio.tu-darmstadt.de

ABSTRACT

We present a massively-parallel implementation of the computation of (co)evolutionary signals from biomolecular sequence alignments based on mutual information (MI) and a normalization procedure to neutral evolution. The MI is computed for two- and three-point correlations within any multiple-sequence alignment. The high computational demand in the normalization procedure is efficiently met by an implementation on Graphics Processing Units (GPUs) using NVIDIA's CUDA framework. GPU computation serves as an enabling technology here insofar as MI normalization is also possible using traditional computational methods [11] but only GPU computation makes MI normalization for sequence analysis feasible in a statistically sufficient sample *and* in acceptable time. In particular, the normalization of the MI for three-point 'cliques' of amino acids or nucleotides requires large sampling numbers in the normalization, that can only be achieved using GPUs.
We illustrate a) the computational efficiency and b) the biological usefulness of two- and three-point MI by an application to the well-known protein calmodulin. Here, we find striking coevolutionary patterns and distinct information on the molecular evolution of this molecule.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming—*Parallel programming*; J.3 [**Life and Medical Sciences**]: Biology and genetics

General Terms

Algorithms, Experimentation, Performance

Keywords

Protein Sequence Analysis, Mutual Information, GPGPU

*to whom correspondence should be addressed

ECMLS'12, June 18, 2012, Delft, The Netherlands.

1. INTRODUCTION

In the post-genome era sequence information is ubiquitous. Large data sets allow the efficient analysis of (co)evolutionary signatures in biomolecules [8, 5, 19], in particular. Such signatures are frequently detected based on the mutual information (MI) from multiple sequence alignments (MSA) of protein or nucleotide sequences. The MI directly reflects the information one position in an MSA contains about another position. However, the high diversity of conservation among the positions in MSAs and finite-size effects in the underlying data sets need to be addressed. Mainly, two routes are taken:

- by 'analytic' normalization, such as the average product correction (APC), the average sum correction (ASC) [6], or the row-column weighting scheme [7] or
- by [9, 26] 'computational normalization', e.g. via Z-scores to a *computational* model of neutral evolution in which columns are shuffled randomly to destroy spurious correlations.

The latter approach induces substantial computational work that can be efficiently dealt with in parallel using Graphics Processing Units (GPUs). A massively-parallel GPU implementation puts the tremendous computational power of modern GPUs into use and lets us benefit from the expected future increases in the degree of parallelism in many-core systems [2].

1.1 Related Work

While general MI computations have been implemented on GPUs before [22, 23], the above mentioned normalization has not been done for coevolutionary biosequence analysis: Shi et. al. [23] used MI for a bioinformatics application.

They investigated generalized correlations in the molecular concentrations of intracellular networks, eventually using fitted kernel techniques to compute MI for real-valued variables, such as concentrations. Furthermore, Shams and Barnes [22] computed MI on images for image comparison. All these studies, however, focused the GPU usage on computing MI, but not on the normalization of such MI to a null-model, which is, however, generally the computationally most expensive part of an MI study. Furthermore, sequence data sets tend to be larger than other data sets (e.g., expression patterns) and almost always need to be normalized because the local Shannon entropies in the residues at a particular site differ by orders of magnitude.

1.2 Our Contribution

Even though the MI computation procedure from one application could in theory be adapted to work on another application as well, the strong constraints massively-parallel architectures like GPUs impose onto the implementation (e.g., very small caches compared to CPUs or the need for control flow to be coherent among threads in a SIMD fashion) make it necessary to design the algorithm from the very beginning for a specific application and to incorporate the characteristics of the problem at hand into the implementation's fine-tuning in order to achieve maximum performance. In our case this has been done for the specific application of histogram-based MI computation on protein or DNA sequences where the number of histogram buckets is comparatively small while on the other hand the whole MI computation needs to be done a huge number of times (in the order of 10^{10} or more for large data sets).

The main contribution of our implementation is the reduction of computation time to the point where MI computation on protein sequences using a computational normalization model with a statistically sufficient number of iterations is usable for practically relevant MSA sizes. Even the computation of three-dimensional mutual information on moderately sized MSAs becomes feasible, potentially paving the way for new research methods: A previously held belief in molecular evolution literature states that typically there is not enough sequence data available to legitimate the use of 3-point MI [25]; however, this problem can be solved with our null model of Sec. 2.2 and our present GPU implementation that addresses the high computational demand of such sampling-based null models.

This paper is organized as follows: In Sec. 2 basic concepts are introduced. Sec. 3 details our GPU implementation. Sec. 4 and 5 evaluate the software, first from an algorithmic (i.e., speed and numerical stability) and then from a bioinformatics perspective. Sec. 6 concludes and points out future work.

2. CENTRAL CONCEPTS

This section introduces central concepts such as mutual information or the shuffling null-model, that define our problem and are needed throughout the rest of the paper.

2.1 Mutual Information

The mutual information (MI) on an MSA is defined for amino acid positions i and j by [17]:

$$\mathrm{MI}_{ij} := \sum_{a,b\in\mathcal{A}} p_{ij}(a,b)\log_2\left(\frac{p_{ij}(a,b)}{p_i(a)\,p_j(b)}\right) \quad (1)$$

where $i,j \in [1;L]$ identify the amino acid positions, $\mathcal{A}$ is the set of symbols (amino acids, nucleotides) at the respective positions and p_{ij}, p_i, p_j are the two- and one-point probabilities[1] to observe a particular amino acid (pairing). MI_{ij} reflects the generalized information between the two columns (i,j) in the MSA. Rewriting the MI [18], one observes $\mathrm{MI}_{ij} = H_i + H_j - H_{ij}$, where H_x are the traditional Shannon entropies H of the single columns i and j, as well as the Shannon entropy of the joint probability distribution for (i,j).

This observation allows us to extend the notion to higher correlation functions, such as a three-column mutual information as

$$\begin{aligned}\mathrm{MI}_{ijk} &= H_i + H_j + H_k - H_{ijk} \\ &= \sum_{a,b,c\in\mathcal{A}} p_{ijk}(a,b,c)\log_2\left(\frac{p_{ijk}(a,b,c)}{p_i(a)\,p_j(b)\,p_k(c)}\right) \quad (2)\end{aligned}$$

MI_{ijk} applied to an MSA reflects then the coevolution among a '3-clique' of amino acid positions.

2.2 Null-Model

The overall magnitude or scale of the signal one can observe in the mutual information values of Eqs. 1 and 2 is largely determined by the single-column entropies H_i, H_j, and H_k, respectively. To normalize to a shared reference scale for all columns in an MSA regardless of the single column entropies H_i, a null model of neutral evolution was suggested [9, 11]. Here, one shuffles the columns i, j (and k) independently n_{iter} times, recomputes the $\widetilde{\mathrm{MI}}_{ij}$ or $\widetilde{\mathrm{MI}}_{ijk}$ after each shuffling, and derives an empirical distribution for the $\widetilde{\mathrm{MI}}_{ij}$ or $\widetilde{\mathrm{MI}}_{ijk}$ values of the shuffled MSAs. The shuffling destroys residual correlations, expressed in the H_{ij} and H_{ijk} between the MSA columns, while maintaining the individual column entropies H_i, H_j, and H_k. Assessing the "background noise" and finite-size effects in real-world data [3] suggested that the scaling law of this statistical effect is $\mathcal{O}\left(n_{\mathrm{iter}}^{-1/2}\right)$.

From the empirical distributions of the $\widetilde{\mathrm{MI}}_{ij}$ or $\widetilde{\mathrm{MI}}_{ijk}$ we can derive two major values to compare MI_{ij} with $\mathrm{MI}_{i'j'}$ for different pairs (i,j) and (i',j'), or MI_{ijk} with $\mathrm{MI}_{i'j'k'}$, respectively: the Z-scores Z_{ij} and Z_{ijk} and the percentiles c_{ij} and c_{ijk} of MI_{ij} with respect to $\widetilde{\mathrm{MI}}_{ij}$ or of MI_{ijk} with respect to $\widetilde{\mathrm{MI}}_{ijk}$. Here, c_{ij} is defined as the fraction of shuffled $\widetilde{\mathrm{MI}}_{ij}$ smaller than the MI_{ij} for the original MSA. Thus, a value of $c_{ij} = 1$ indicates that all shuffled columns (i,j) showed a smaller MI than the original one, indicating a significant coevoultion on the background of neutral evolution. Note, that the same definition holds for c_{ijk}.

Furthermore, for the Z-score normalization with

$$Z_{ij} := \frac{\mathrm{MI}_{ij} - \langle\widetilde{\mathrm{MI}}_{ij}\rangle}{\sigma\left(\widetilde{\mathrm{MI}}_{ij}\right)} \quad (3)$$

we need to compute two expectation values: the average of the 'shuffled' MI $\langle\widetilde{\mathrm{MI}}_{ij}\rangle$ and its variance $\sigma\left(\widetilde{\mathrm{MI}}_{ij}\right)$. Using the 'trick' $\sigma\left(\widetilde{\mathrm{MI}}_{ij}\right)^2 = \langle\widetilde{\mathrm{MI}}_{ij}^2\rangle - \langle\widetilde{\mathrm{MI}}_{ij}\rangle^2$ (with $\langle\dots\rangle$ denoting expectation values) we can do this in linear time within constant memory with respect to the number of sequences in

[1] empirically obtained from histograms as relative counts

the MSA. If the percentiles c_{ij} or c_{ijk} need to be computed, this is done in a separate step.

2.3 Diagonal Entries in MI Tensors as Reference

The objects MI_{ij}, Z_{ij}, MI_{ijk}, and Z_{ijk} are 2- and 3-dimensional tensors[2], respectively. The diagonal elements MI_{ii} and MI_{iii} serve as a reference for the underlying sequence Shannon entropy, which follows from the definitions in Eqs. 1 and 2:

$$\begin{aligned} \mathrm{MI}_{ii} &= H_i + H_i - \underbrace{H_{ii}}_{=H_i} \\ &= 2H_i - H_i \\ &= H_i \qquad (4) \\ \mathrm{MI}_{iii} &= H_i + H_i + H_i - \underbrace{H_{iii}}_{=H_i} \\ &= 3H_i - H_i \\ &= 2H_i \qquad (5) \end{aligned}$$

These entries are therefore a reference on the (local) sequence variability of an MSA. In fact, the aforementioned analytic normalization procedures [6, 7] make heavy use of this property. Furthermore, the respective diagonal entries of the Z tensors vanish altogether due to the implemented null model of shuffled MSA columns, which maintain the column-wise entropies H_i (see Sec. 2.2).

This property of MI_{ij}, Z_{ij}, MI_{ijk}, and Z_{ijk} will be used to discuss and gauge the relation between the two- and three-point MI analysis below.

3. IMPLEMENTATION

In this section the computation process is mostly described for the case of two-point MI. Three-point MI computation works analogously with one more dimension (e.g., the matrices being computed are not matrices but general tensors). Also, it is first described for the sequential single-core case and then the parallelization scheme is pointed out.

3.1 Sequential Version

The computation process consists of two major parts: In the first step (see Listing 1) the MI matrix, which contains the MI_{ij} values for every combination of MSA column pairs i and j, is computed according to Eq. 1. Note, that it suffices to compute the matrix' lower triangle (or one sixth of the whole tensor in case of 3-point MI), because the matrix is symmetric (from Eq. 1 follows that $\mathrm{MI}_{ij} = \mathrm{MI}_{ji}$). Also note, that the one-point probabilities $p_i(x)$ are precomputed for all columns i and characters x. This computation is linear in the number of columns as opposed to the quadratic complexity of the MI matrix computation. The $p_i(x)$ are then stored in the sequence alignment data structure and accessed via MSA.getOnePointProbability(...).

In the second step (see Listing 2) the $\left(\langle\widetilde{\mathrm{MI}}_{ij}\rangle\right)$ and the $\left(\langle\widetilde{\mathrm{MI}}^2_{ij}\rangle\right)$ matrix, which are needed for normalizing the MI (see Eq. 3), are computed. This is done by alternately shuffling all columns in the MSA independently (e.g., by using a Knuth's shuffling [14, Sec. 3.4.2] on all elements of the respective column) and then computing the MI on the shuffled MSA. Again it is sufficient to compute the matrices' lower triangle. Also, the $p_i(x)$ can be precomputed once before the whole shuffling null-model computation, because shuffling a column does not change its one-point character distribution. The whole process of shuffling and MI computation is repeated for a statistically sufficient number of times, typically around 10,000 iterations.

[2]Note, that a two-dimensional tensor is identified with a traditional matrix here.

```
float computeMI(SequenceAlignm& MSA, int i, int j) {
  int histogram[ALPHABETSIZE][ALPHABETSIZE];
  memset(histogram, 0, sizeof(twoOrThreePointOccs));
      // fill with 0s
  for each sequence in MSA
    histogram[MSA.getChar(sequence, i)][MSA.getChar(
        sequence, j)]++;
  float MI = 0;
  for each x in alphabet {
    float p_i_x = MSA.getOnePointProbability(x, i);
    if (p_i_x < EPSILON) continue;
    for each y in alphabet {
      float p_j_y = MSA.getOnePointProbability(y, j);
      if (p_j_y < EPSILON) continue;
      float p_ij_xy = float(histogram[x][y]) / float(
          numberOfSequences);
      MI += p_ij_xy * log2(p_ij_xy / (p_i_x * p_j_y));
    }
  }
  return MI;
}

void computeMIMatrix(SequenceAlignm& MSA, Matrix&
    MI) {
  for each column i in MSA
    for each column j >= i in MSA
      MI.set(i, j, computeMI(MSA, i, j);
}
```

Listing 1: Sequential computation of the MI matrix

```
void computeEMIMatrices(SequenceAlignm& MSA,
    Matrix& EMI, Matrix& EMI2) {
  for it = 1 to numIterations
    for each column i in MSA
      MSA.randomlyPermuteAllElementsInColumn(i);
    for each column i in MSA
      for each column j >= i in MSA
        float MI = computeMI(MSA, i, j);
        EMI.set(i, j, EMI.get(i, j) + MI);
        EMI2.set(i, j, EMI2.get(i, j) + MI*MI);
  EMI.divideEveryElementBy(numIterations);
  EMI2.divideEveryElementBy(numIterations);
}
```

Listing 2: Sequential computation of the $\left(\langle\widetilde{\mathrm{MI}}_{ij}\rangle\right)$ and the $\left(\langle\widetilde{\mathrm{MI}}^2_{ij}\rangle\right)$ matrix

3.2 Parallelization

The nested loops in the sequential code (iteration over all i, j, x and y) reveal a lot of potential for parallelization: Each matrix entry can be computed independently of all others and as the $\left(\langle\widetilde{\mathrm{MI}}_{ij}\rangle\right)$ and $\left(\langle\widetilde{\mathrm{MI}}^2_{ij}\rangle\right)$ matrices are typically very large (their size is quadratic in the MSA's sequence length L) there is enough inherent parallelism for not

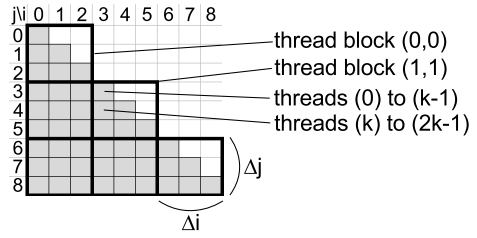


Figure 1: The matrices to be computed are subdivided into submatrices of $\Delta i \cdot \Delta j$ elements and the content of each submatrix is computed by one block of $k \cdot \Delta i \cdot \Delta j$ threads.

only parallelizing it on a multi-core system but rather on a many-core system capable of running hundreds of thousands of threads.

We parallelized the computation by dividing the $\left(\langle \widetilde{\mathrm{MI}}_{ij} \rangle\right)$ and the $\left(\langle \widetilde{\mathrm{MI}}^2_{ij} \rangle\right)$ matrix into small submatrices of $\Delta i \cdot \Delta j$ elements and assigning the computation of each submatrix to one CUDA block of threads [1], as can be seen in Fig. 1. The reason for this subdivision is the following: Each thread block computes $\Delta i \cdot \Delta j$ matrix entries, but for computing these entries it only needs to access $\Delta i + \Delta j$ MSA columns. As cache size is typically very limited in GPUs (currently 16 KB per block [1]), this subdivision is necessary to let the accessed MSA columns fit completely into the cache.

Inside each submatrix, groups of k threads work together on the computation of one matrix entry, leading to $k \cdot \Delta i \cdot \Delta j$ threads per block. This parallelization is extremely fine-grained, as is necessary for efficient GPU computation.

At first sight the work on the main diagonal blocks seems to be unbalanced, because not all entries in those blocks need to be computed (white boxes in the main diagonal blocks in Fig. 1), dooming some threads to idleness. We found this to be negligible, mainly because the number of blocks on the main diagonal is only linear in the sequence length L, as opposed to the quadratic total number of blocks.

In order to compute one matrix entry, each group of k threads builds a shared histogram with $|\mathcal{A}|^2$ ($|\mathcal{A}|$ being the size of the amino acid or nucleotide alphabet) entries to compute the $p_{ij}(a, b)$ for Eq. 1. Usually one would use basic integers for this histogram. Shared memory, that is local to thread blocks and much faster accessible than the global RAM, is almost as limited as memory cache (currently 48 KB per block [1]). Thus, in cases where we know that histogram entries will not exceed a certain limit using basic integers seems wasteful. Therefore, we pack groups of up to four histogram entries into one integer using bit-masking and shifting. E.g., if the MSA at hand consists of 1023 or less sequences, no histogram entry can exceed 1023, 10 bit suffice for each entry and 3 entries can be packed into one 32 bit integer.

The compression factors as well as other parameters like Δi and Δj are dependent on the problem size (the MSA's sequence length and number of sequences, but also whether 2-point or 3-point MI is to be computed or whether the underlying alphabet is the amino acid or the nucleotide alphabet). As standard C++ templating can be used in CUDA kernels as well, we implemented all these parameters as template parameters for the GPU kernel and tuned each template parameter individually for each problem size. Histogram compression in combination with templating reduced shared memory consumption and increased overall performance considerably.

The shuffling, which is executed in alternation with the MI computation, turned out to be very time-consuming as well. Thus, we used a highly GPU suitable algorithm: the butterfly network [16, Sec. 3.2], which is usually used in network routing or in Fast Fourier Transform (FFT), can be used as a permutation network. Its topology fits the massively-parallel architecture very well because it allows fine-grained parallelism: Each of the parallel exchanges in the permutation network can be executed by a separate thread. Furthermore the memory access pattern is predefined and does not depend on the outcome of previous computations. In [24] we showed the butterfly network to be much faster than other state-of-the-art algorithms and proved that it permutes unbiased if implemented properly.

4. ALGORITHMIC EVALUATION

In this section the implementation is evaluated from an algorithmic perspective. First (Sec. 4.1), it is shown that the numerical error induced by the GPU implementation is negligible because it is overshadowed by the noise inherent to the shuffling null-model. Second (Sec. 4.2), the GPU implementation's speed-up compared to a CPU multi-core version running on a current CPU is shown and discussed.

4.1 Numerical Stability

Fig. 2 a) compares the obtained $\langle \widetilde{\mathrm{MI}}_{ij} \rangle$ and $\langle \widetilde{\mathrm{MI}}^2_{ij} \rangle$ values for an MSA of HIV1-protease computed by a CPU reference implementation with the CUDA implementation. As expected the differences diminish with an increasing number of shuffling null-model iterations. For comparison, Fig. 2 b) gives the difference of two independent CPU runs with different initial random seeds. The almost perfect similarity to Fig. 2 a) shows, that the difference between CPU and GPU is indistinguishable from the problem-inherent noise imposed by the algorithm's random nature and is thus unavoidable. The GPU-induced noise, that originates from a different floating-point hardware implementation as well as a different evaluation order of arithmetic expressions due to the parallel algorithm, seems to be insignificant.

Additionally, we verified the implementation's correctness by a comparison with an independent package [11] written by a different programmer in a different language (`R` and `C`) and obtained similar results regarding numerical stability.

Another information that may be deduced from Fig. 2 a) is the number of shuffling null-model iterations one approximately needs to obtain sufficiently accurate results. In our case $10,000$ iterations seem to be sufficient. This is actually a large number — greater than the (resource restricted) number of iterations frequently employed in previous studies [27, 9] that relied on CPU implementations. This supports our notion, that massively-parallel implementations such as ours are needed to extend the information theoretical analysis of molecular evolution to the genome scale. We discuss this property further in the next section.

4.2 Computational Efficiency

The massively-parallel implementation's advantage is obviously the reduction of wall clock time for computing the full $\left(\langle \widetilde{\mathrm{MI}}_{ij} \rangle\right)$ and $\left(\langle \widetilde{\mathrm{MI}}^2_{ij} \rangle\right)$ matrices for all $(i, j) \in [1 \dots L]^2$.

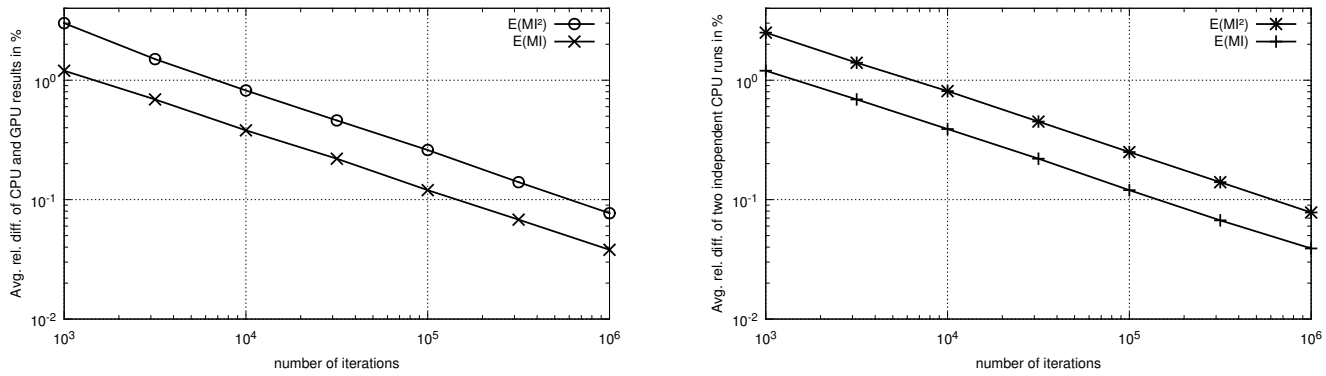


Figure 2: Average relative difference between the matrices $(\langle\widetilde{\mathrm{MI}}_{ij}\rangle)$ and $(\langle\widetilde{\mathrm{MI}}^2_{ij}\rangle)$ generated by a) the GPU and the CPU implementation and b) two independent runs of the CPU implementation. Note the strong similarity to the CPU-GPU comparison.

n_{iter}	Core i7-960 @ 3.2 GHz, 4 threads	GeForce GTX 480	relative speed-up
calmodulin (753 sequences of length 264 aas), 2-point MI			
10,000	12.7 min	1.03 min	12.3x
calmodulin (753 sequences of length 264 aas), 3-point MI			
10,000	3.09 days	1.15 days	2.7x
ribozymes (140 seqs. of length 71 nts), 3-point MI			
10,000	12.5 min	0.445 min	28.1x

Table 1: Comparison of wall times between the CUDA implementation of the shuffling null-model of Eq. 3 and the equivalent CPU implementation, using all 4 CPU cores; lengths and alphabet sizes vary (aas=amino acids $\to 22$, nts=nucleotides $\to 6$, including a gap character '-' and a not-known-character 'X').

Wall clock time comparisons are shown in Table 1. For the CPU timings the CPU code was parallelized to 4 threads (the Core i7-960 has 4 physical cores) using OpenMP. 8 threads have been tried as well, but they turned out to be slower than 4 threads.

As can be seen from Table 1 speed-ups compared to the multi-core implementation are highly problem size dependent and range from roughly 3 to more than 20.

There are multiple reasons for the problem size dependent speed-up: First, if the number of sequences is high, the maximum histogram entry, that can be expected, is large and histogram compression can only safely be used with a small compression factor or no compression at all (see histogram compression in Sec. 3.2), leading to increased shared memory consumption. Also, if 3-point MI is to be computed, the histogram's size is $|\mathcal{A}|^{\mathbf{3}}$, which increases shared memory consumption drastically.

Theoretically, each block of threads (see Fig. 1) has 48 KB of fast shared memory available, but practically it is advisable to use only a fraction of this, because this enables the GPU scheduler to map multiple blocks onto one multiprocessor and switch between blocks when one block is busy with waiting for data from the memory [1]. If histogram compression cannot be used (e.g., number of sequences $> 2^{16}$) or 3-point is computed, this may not be possible anymore, significantly reducing the GPU implementation's performance.

Another issue are small data sets: If the sequence length is very short, the number of blocks is small, which also makes switching between blocks inside one multi-processor harder. If the number of sequences is very short, the histogram generation itself, whose speed-up is almost linear in the number of threads, is done very fast, which in turn makes other parts with sublinear speed-up (e.g., parallel reduction that sums over all x and all y in the MI formula, or the shuffling process) more relevant in the overall timing.

The highest speed-up has been achieved for a nucleotide data set, where the alphabet (and thus the histogram as well) is considerably smaller. Also, caching during reading the precomputed one-point probabilities $p_i(x)$ is much more effective. This is not the standard case and the 12.3 speed-up for the protein sequence set is more representative for typical use cases.

Concerning speed-up we remark that, while the theoretic peak performances (roughly 50 GFlops (depending on the clock frequency) for the i7-960 and 1345 GFlops for the GTX 480) allow a maximum speed-up of about 27 for realistic, compute intensive (as opposed to memory-intensive) applications such as the MI computation, the constraints the hardware imposes onto the programmer (e.g., coherence among concurrently running threads) only allow much lower speed-ups [10, 15]. Our own experiments showed, that even numerical NVIDIA code (namely the cublas project), that can be regarded as highly optimized, runs at performances much lower than the theoretical peak performance: 765 GFlops. Extremely high speed-ups need to be seen skeptical, as they are likely to result from sub-optimal CPU implementations or implementation 'tricks' that have been exploited on the GPU but not on the CPU (as has indeed been done with the histogram compression, which is one of the reasons for the high speed-up of the ribozyme file).

5. APPLICATION TO CALMODULIN

Calmodulin is a small, acid- and heat-stable protein which consists of approximately 149 amino acid residues. Its sequence is highly conserved in all eukaryotes. Calmodulin has no enzymatic activity on its own, but binds Ca^{2+}-ions through EF-hand motifs which leads to a conformational change within the protein. Thereby, calmodulin can interact and regulate a variety of proteins such as phosphodiesterase, brain adenylate cyclase, myosin light chain kinases, brain membrane kinases, calcineurin and nitric oxide synthase. Furthermore, calmodulin is a multifunctional, ubiquitous second messenger protein and plays an important role as a mediator in many cell signaling pathways. Due these biological functions and due to its broad interaction specificity and the regulatory function in response to intra-cellular Ca^{2+}-concentration, we expect calmodulin to be subject to distinct and complex selective pressures in its molecular evolution in eukaryotes.

5.1 Data Preparation

A BLAST-search based on a sequence of a human calmodulin (149 amino acids) was performed on the NCBI-Server[3]. Approximately $1,206$ sequences with an sufficiently low E-value below 10^{-50} were chosen and aligned in a multiple sequence alignment using the alignment tool CLUSTAL-W.

We obtained an alignment with length 264 for the $1,206$ sequences, which were manually curated to avoid duplications and other artifacts in the sequence data; resulting in a final sequence set of 753 sequences.

5.2 Results for 2- and 3-point MIs

First, using the visual analytics tool `MatrixViz` [4] we investigated the classical two-point MI of Eq. 1 for patterns and connections of real MI values and the normalized Z-scores.

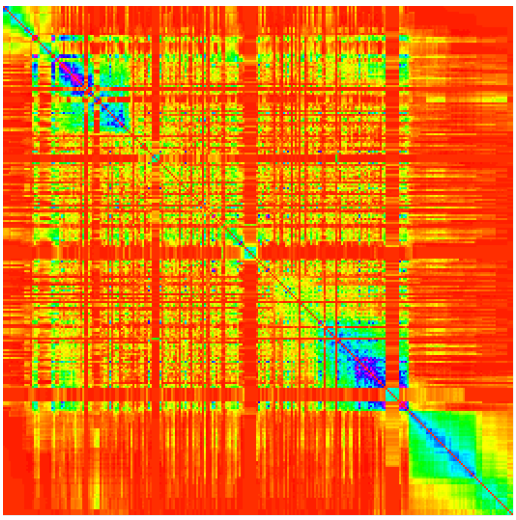
Figure 3 shows the results for the two-point mutual information calculations and the normalization to Z-scores in the null model of Section 2.2. Most importantly, we notice a correspondence to the secondary structure of the protein, in particular significant coevolution occurs at the N- and C-terminus domains.

As previously observed [9], conserved residues induce 'stripes' of low MI values in the 2D visualization. The extended region in the middle of the protein backbone stretching from residue numbers 40 to 110 corresponds to the 'connecting' α-helix (compare Figure 6) This linker is rather conserved as is obvious in the entropy shown in the lower portion of Fig. 3 (b). Therefore, the coevolution of residues in this isolated secondary structure is non-surprisingly rather low – the helix is just maintained.

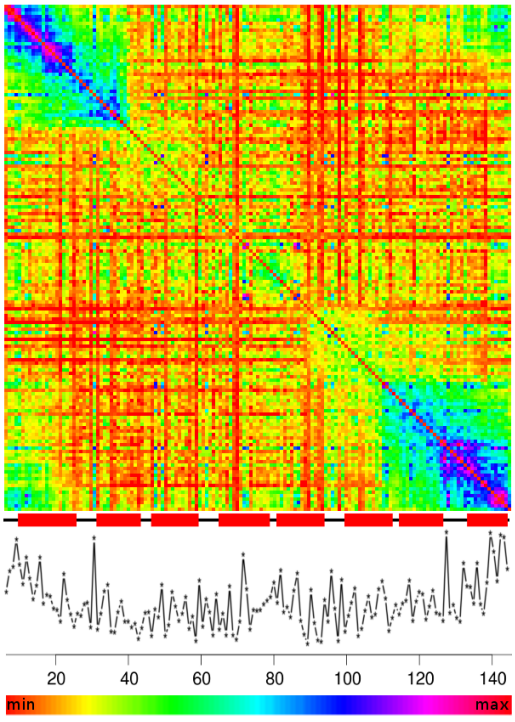
5.3 Are 3-Point MIs Simple Constructs of 2-Point MIs?

The 3-point MI_{ijk} of Eq. 2 and its normalization to Z-scores or percentiles may reveal more involved and complex coevolution than the 2-point MI. Theoretically, the 3-point MI_{ijk} might be just given by the coevolution of pairs (i,j), (i,k), or (i,k), alone. Or, in contrast, MI_{ijk} might reveal unique new insights beyond what one can deduce from the 2-point MI analysis of Sec. 5.2. Here, we show, that this is

[3] `http://www.ncbi.nlm.nih.gov`



(a)



(b)

Figure 3: (a) color-coded Z values of Eq. 3 for the 264 positions in the MSA of calmodulin (blue=high values, red=low values of MI). (b) same as above, but restricted to those resiudes present in the human variant of calmodulin with $L = 149$ amino acids; we also depict the secondary structure annotation (red blocks=helices) of the pdb-file 2W73-B of calmodulin; secondary structure information was obtained using the software `yasara` [13]. Furthermore we indicate the local sequence entropy H_i for each position i in the MSA, computed with the `BioPhysConnectoR` package [11] implemented in `R` [21]. Pictures were rendered and analyzed with the `MatrixViz` package for (2d) MI analysis [4].

indeed the case and that the 3-point MI (and its normalization by the null model) is an interesting object on its own.

In Fig. 4 we show a scatter plot of the MI values for the three-amino-acid-clique (i, j, k) and combinations of the respective two-point MIs for the pairs (i, j), (i, k), and (j, k) constituting the clique (i, j, k).

Using the relation of diagonal entries MI_{iii} from Eq. 5, we can immediately see that the straight line in Fig. 4 consists of the diagonal entries of the two- and three-dimensional tensors MI_{ij} and MI_{ijk}. This line can serve as a reference of trivial and obvious correspondence of 2-point and 3-point MIs for the special case of diagonal entries (i, i, i).

Fig. 4 (a) suggests that for some cliques (i, j, k) the MI_{ijk} has a different behavior than the combination of the respective 2-point MIs. This suggests a potential contradiction to previous assumptions [20] on the 'composability' of MIs and models on effective two-body interactions in intra-molecular coevolution. Note, that it is not necessary to show two distinct clusters in Fig. 4, a bimodal distribution distinct from the diagonal is already sufficient to disprove the above mentioned assumption on composability.

To investigate this further, we can now employ the computationally efficient normalization to the null model of neutral evolution in our GPU implementation and obtain, e.g., the percentiles c_{ij} and c_{ijk}.

In Fig. 5 we show an analysis of these normalized values. In particular Fig. 5 (a) suggests a binary classification of cliques (i, j, k): those for which c_{ijk} is larger than $\max(c_{ij}, c_{ik}, c_{jk})$ and those for which the opposite holds. For those cliques (i, j, k) of the first case we define the set $\mathcal{C}$ as follows

$$\mathcal{C} := \{(i, j, k) | \max(c_{ij}, c_{ik}, c_{jk}) \leq c_{ijk}\} \tag{6}$$

Members of $\mathcal{C}$ are those 3-amino acid cliques for which there is more coevolution among observable the three simultaneously than what one can maximally detect in any combination of two of them (i, j), (i, k), or (j, k). Thus, for those cliques in $\mathcal{C}$ the 3-point MI_{ijk} contribute new and unique insight into the molecular coevolution.

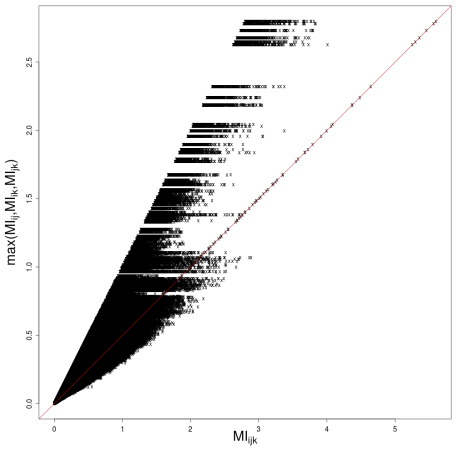
Our results clearly show that the MI for high-order correlations (at least three residues involved) is not decomposable into simpler (i.e., 2-point) MI contributions. Thus, cliques of three coevolving residues seem to coevolve in a much more involved fashion than was claimed previously [25].

For a biological interpretation of these interesting patterns in the set $\mathcal{C}$ we counted the number of cases in which a residue i contributed to this set. In Fig. 6 we overlay this as a color code onto the 3D-structure of the protein.

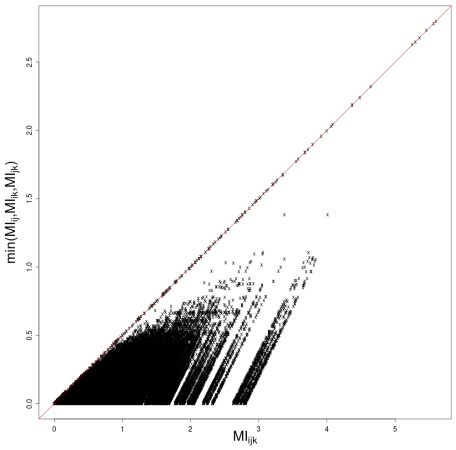
Most interestingly, we found a correlation to structural *and* functional characteristics of calmodulin. In particular, Table 2 shows almost all residues to be either members of the flexible helices C, E, and F or involved in Ca^{2+} binding. Furthermore, the residues in $\mathcal{C}$ that are found in helices appear at the start or end of the respective helix, suggesting a distinct contribution to the maintenance of this structural element. Furthermore, the helices involved in the conformational change are found also.

Now, conformational changes and binding of cofactors are typically synergistic process; a property we found to be present significantly enriched in the molecular coevolution of 3-cliques[4] in comparison to 2-point MIs.

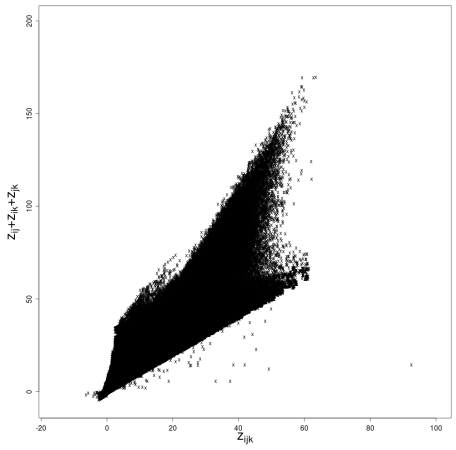
[4]and potentially beyond, e.g., in 4-, 5-, ... generalized MIs



(a)

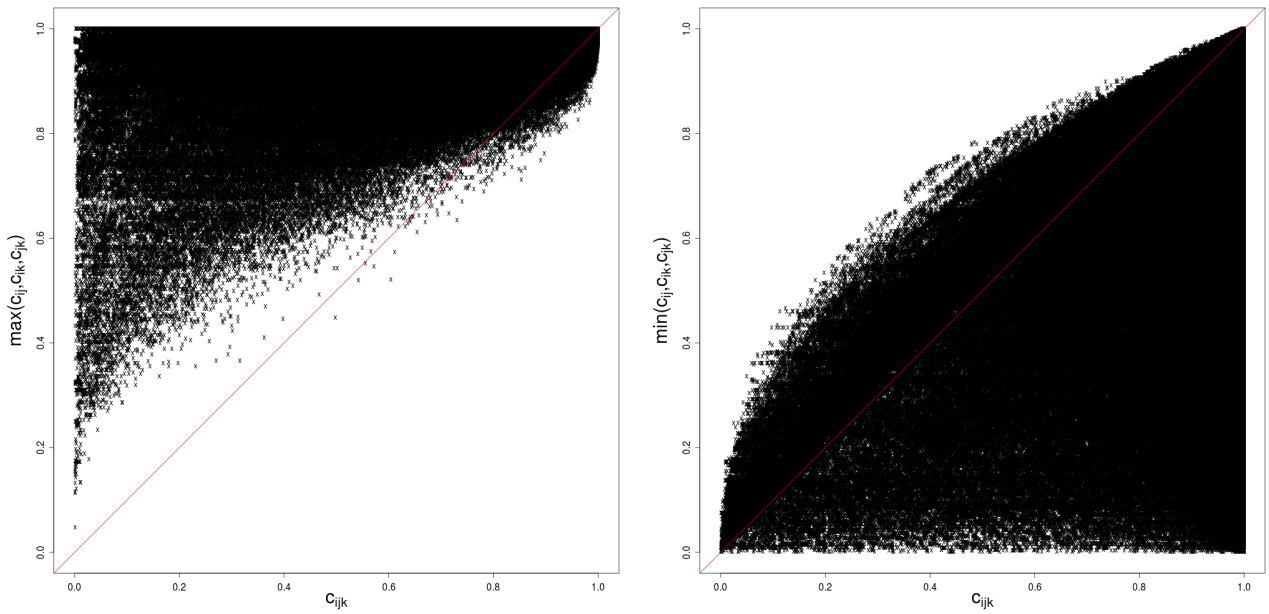


(b)



(c)

Figure 4: (a) comparison of the maximum of the 2-point MI in a 'clique' $\max(\mathrm{MI}_{ij}, \mathrm{MI}_{ik}, \mathrm{MI}_{jk})$ **on the** y**-axis with the 3-point** MI_{ijk} **of the 'clique'** (i, j, k) **on the** x**-axis, the red line is a straight line with slope** $1/2$ **(see text and Eq. 5 for explanation); (b) same as above, but for** $\min(\mathrm{MI}_{ij}, \mathrm{MI}_{ik}, \mathrm{MI}_{jk})$ **on the** y**-axis; (c) the same as in (a), but for the** Z**-scores of Eq. 3**



(a) (b)

Figure 5: Comparing combinations of normalized MI of two-point coevolutionary signals and 'clique'-like combinations of amino acids (i, j, k). Here, the normalization or reference values are the percentiles c_{ijk} and c_{ij}, c_{ik}, and c_{jk}. The red line indicates equality between the plotted values.

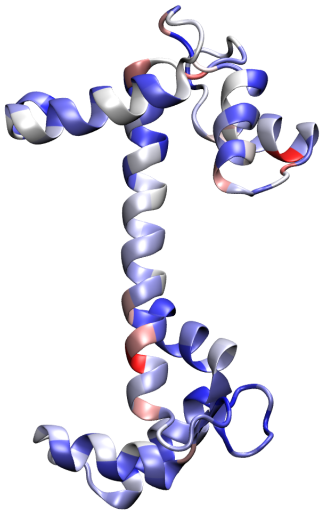


Figure 6: The three-dimensional structure of the calmodulin protein. The color (red=high values, blue=low values) indicates the frequency with which an amino acid contributed to the 3-'clique's set $\mathcal{C}$ of Eq. 6. The structure was rendered with `VMD` [12].

residue	no.	2^{nd} structure	helix	function
G	26	C		Ca^{2+}-binding
I	28	C		Ca^{2+}-binding
L	40	C		
P	44	C		
E	48	H	C	
I	53	H	C	
E	55	H	C	
I	64	C		Ca^{2+}-binding
F	66	C		Ca^{2+}-binding
E	68	H	D	
E	85	H	E	
E	88	H	E	
A	89	H	E	
F	93	C		Ca^{2+}-binding
S	102	H	F	Ca^{2+}-binding

Table 2: The residues in the set $\mathcal{C}$ appearing most frequently. The secondary structure elements are abbreviated: H=helix, C=coil. The functional annotation is based on [28].

6. CONCLUSIONS

In the present study we have obtained interesting insights into the molecular (co)evolution of an important regulatory protein (calmodulin). We have shown that distinct patterns of coevolution can be found. We generalized from the coevolution of two amino acids among each other, to 'cliques' of three residues potentially coevolving in a more involved fashion than previously assumed. It was found that indeed such higher-order correlations are present and connected to function and dynamo-structural properties of calmodulin, showing a pressing need for efficient computational tools to incorporate higher-order terms in future research in molecular evolution.

Generally speaking, the GPU times are much faster than what is achievable with a current multi-core CPU. MI com-

putation and normalization on moderately sized MSAs can now be computed 3 to 28 times faster. As has been argued in Sec. 4 one order of magnitude can be seen as respectable and realistic given current hardware. Also, this speed-up has an important consequence: It allows quasi-interactive normalized MI computation. If some parameter needs to be changed in the tool chain in a step before the MI computation (e.g., the alignment process) this can be done and the MI can quickly be recomputed and normalized, which allows to work with coevolutionary sequence analysis in a completely different way.

One may argue, that the degree of parallelism also increases in CPUs in the future. This, however, holds even more for future GPUs. Currently there is a strong trend from multi-core towards many-core systems, which can be expected to continue in the near future [2]. With an efficient GPU implementation we are on a good way to benefit from this development.

3-point MI computationtimes still need improvement. But even though the 2.7x speed-up for the calmodulin 3-point MI evaluation from Table 1 may seem small, it is faster for the ribozymes file (which is doubtlessly not the most relevant use case) as well as for other files we tested during our testing phase. In any case the GPU implementation acts as an enabling technology by reducing the run-time to a point where one actually starts to think about using 3-point MI. There is a fundamental difference between ~ 3 days and ~ 1 day for a computational biologists' sequence analysis workflow and we argue that 3-point MI computation thus comes into reach as a practicable tool for future investigation.

We found interesting patterns of higher-order coevolution, in particular in important structural motifs and functional sites. It remains to be seen, whether this holds for other proteins. However, with the present software such large-scale investigations on a comprehensive set of proteins, potentially covering the known protein-space, have become possible due to the speed-ups made possible by GPU-usage.

6.1 Future Work

For future investigation it might be interesting to look into auto-tuning. The GPU kernel's performance heavily depends on a lot of template parameters which all depend on the problem size. Also they can only be tuned to a very small set of GPUs by the programmer and for other, possibly yet-to-appear GPUs the parameters may not be the same. Therefore it may be very rewarding to let these parameters be automatically tuned.

The parallelization of software to hundreds of thousands of threads on a single GPU raises the question whether it is also parallelizable on a higher level for multi-GPU support. We are therefore currently parallelizing the implementation for a cluster with multiple GPUs per cluster node. Here, one iteration of the shuffling null-model is computed by one GPU (with different random generator seeds per GPU) and iterations are distributed to the GPUs in a round-robin fashion. This can be done with an almost arbitrary number of GPUs, because each iteration is independent of all other iterations. First experiments showed, that this parallelization scheme yields a speed-up that is almost linear in the number of GPUs, mainly because the whole task is compute intensive and network traffic (sending the MSA to the nodes and sending the matrices back to the master node) is negligible compared to the amount of computation. On a 32 GPU cluster we achieved a speed-up of almost 32 on top of the basic single-GPU speed-up.

7. ACKNOWLEDGMENTS

This work was partly supported by the DFG Graduate School 1657 and DFG grant no. HA 5261/3-1. We thank Franziska Hoffgaard for providing the ribozyme MSA for runtime benchmarks of Table 1.

APPENDIX

A. AVAILABILITY

The source code to our software as well as installation and usage documentation can be obtained at `http://www.gris.tu-darmstadt.de/projects/comic/` and may be used and redistributed under the terms of the GPLv3 license; provided that this study is cited in all publications and other work using results obtained with it.

B. REFERENCES

[1] CUDA C Programming Guide. `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`, accessed on 17-April-2012.

[2] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Dept., UC Berkeley, Dec 2006.

[3] P. Boba, P. Weil, F. Hoffgaard, and K. Hamacher. Intra- and inter-molecular coevolution: The case of HIV1 protease and reverse transcriptase. In A. Fred, J. Filipe, and H. Gamboa, editors, *Biomedical Engineering Systems and Technologies*, volume 127 of *Springer Communications in Computer and Information Science*, pages 356–366. Springer, Berlin Heidelberg, 2011.

[4] S. Bremm, T. Schreck, P. Boba, S. Held, and K. Hamacher. Computing and visually analyzing mutual information in molecular co-evolution. *BMC Bioinformatics*, 11:330, 2010.

[5] F. M. Codoñer and M. A. Fares. Why should we care about molecular coevolution? *Evol Bioinform Online*, 4:29–38, Jan 2008.

[6] S. D. Dunn, L. M. Wahl, and G. B. Gloor. Mutual information without the influence of phylogeny or entropy dramatically improves residue contact prediction. *Bioinformatics*, 24(3):333–40, Feb 2008.

[7] R. Gouveia-Oliveira and A. G. Pedersen. Finding coevolving amino acid residues using row and column weighting of mutual information and multi-dimensional amino acid representation. *Algorithms Mol Biol*, 2(1):12, Jan 2007.

[8] R. Gouveia-Oliveira, F. S. Roque, R. Wernersson, T. Sicheritz-Ponten, P. W. Sackett, A. Molgaard, and A. G. Pedersen. InterMap3D: predicting and visualizing co-evolving protein residues. *Bioinformatics*, 25(15):1963–1965, Aug 2009.

[9] K. Hamacher. Relating sequence evolution of HIV1-protease to its underlying molecular mechanics. *Gene*, 422:30–36, 2008.

[10] D. F. Harlacher, S. Roller, F. Hindenlang, C.-D. Munz, T. Kraus, M. Fischer, K. Geurts, M. Meinke, T. Klühspies, V. Metsch, and K. Benkert. Highly efficient and scalable software for the simulation of turbulent flows in complex geometries. In W. E. Nagel, D. B. Kröner, and M. M. Resch, editors, *High Performance Computing in Science and Engineering '11*, pages 289–307. Springer Berlin Heidelberg, 2012.

[11] F. Hoffgaard, P. Weil, and K. Hamacher. BioPhysConnectoR: Connecting sequence information and biophysical models. *BMC Bioinformatics*, 11(1):199, 2010.

[12] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *J. Mol. Graphics*, 14:33–38, 1996.

[13] D. Jones. Protein secondary structure prediction based on position-specific scoring matrices. *J. Mol. Biol.*, 292:195–202, 1999.

[14] D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.)*. 1997.

[15] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. *SIGARCH Comput. Archit. News*, 38(3):451–460, June 2010.

[16] F. Leighton. *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes*. Number 1. M. Kaufmann Publishers, 1992.

[17] O. Lund, M. Nielsen, C. Lundegaard, and C. K. S. Brunak. *Immunological Bioinformatics*. MIT Press, Cambridge, 2005.

[18] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge, 2. edition, 2004.

[19] L. C. Martin, G. B. Gloor, S. D. Dunn, and L. M. Wahl. Using information theory to search for co-evolving residues in proteins. *Bioinformatics*, 21(22):4116–4124, Nov 2005.

[20] F. Morcos, A. Pagnani, B. Lunt, A. Bertolino, D. S. Marks, C. Sander, R. Zecchina, J. N. Onuchic, T. Hwa, and M. Weigt. Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proc. Nat. Acad. Sci.*, 108(49):E1293–E1301, 2011.

[21] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[22] R. Shams and N. Barnes. Speeding up mutual information computation using NVIDIA CUDA hardware. In *Digital Image Computing Techniques and Applications, 9th Biennial Conference of the Australian Pattern Recognition Society on*, pages 555 –560, Dec 2007.

[23] H. Shi, B. Schmidt, W. Liu, and W. Müller-Wittig. Parallel mutual information estimation for inferring gene regulatory networks on GPUs. *BMC Research Notes*, 4(1):189, 2011.

[24] M. Waechter, K. Hamacher, F. Hoffgaard, S. Widmer, and M. Goesele. Is your permutation algorithm unbiased for $n \neq 2^m$? In *9th Int. Conf. on Parallel Processing & Appl. Math. (PPAM2011)*, Sept 2011.

[25] M. Weigt, R. A. White, H. Szurmant, J. A. Hoch, and T. Hwa. Identification of direct residue contacts in protein-protein interaction by message passing. *P Natl Acad Sci Usa*, 106(1):67–72, Jan 2009.

[26] P. Weil, F. Hoffgaard, and K. Hamacher. Estimating sufficient statistics in co-evolutionary analysis by mutual information. *Computational Biology and Chemistry*, 33(6):440–444, 2009.

[27] R. A. White, H. Szurmant, J. A. Hoch, and T. Hwa. Features of protein-protein interactions in two-component signaling deduced from genomic libraries. *Methods Enzymol.*, 422:75–101, 2007.

[28] M. Zhang, T. Tanaka, and M. Ikura. Calcium-induced conformational transition revealed by the solution structure of apo calmodulin. *Nat Struct Mol Biol*, 2(9):758–767, 1995.