Reconfigurable Hardware Computing for Accelerating Protein Folding Simulations using the Harmony Search Algorithm and the 3D-HP-Side Chain Model *

César Manuel Vargas Benítez, Marlon Scalabrin,
Heitor Silvério Lopes, Carlos R. Erig Lima

Bioinformatics Laboratory, Federal University of Technology - Paraná,
Av. 7 de setembro, 3165 80230-901, Curitiba (PR), Brazil
cesarvargasb@gmail.com, marlonscalabrin@yahoo.com.br,
hslopes@utfpr.edu.br, erig@utfpr.edu.br

1 Introduction

Proteins are essentials to life and they have countless biological functions. They are synthesized in the ribosome of cells following a template given by the messenger RNA (mRNA). During the synthesis, the protein folds into an unique three-dimensional structure, known as native conformation. This process is called protein folding. Several diseases are believed to be result of the accumulation of ill-formed proteins.Therefore, understanding the folding process can lead to important medical advancements and development of new drugs.

Thanks to the several genome sequencing projects being conducted in the world, a large number of new proteins have been discovered. However, only a small number of such proteins have its three-dimensional structure known. For instance, the UniProtKB/TrEMBL repository of protein sequences has currently around 16.5 million records (as in july/2011), and the Protein Data Bank – PDB has the structure of only 74,800 proteins. This fact is due to the cost and difficulty of unveiling the structure of proteins, from the biochemical point of view. Computer Science has an important role here, proposing models and computation approaches for studying the Protein Folding Problem (PFP).

The Protein Folding Problem (PFP) can be defined as finding the three-dimensional structure of a protein by using only the information about its primary structure (e.g. polypeptide chain or linear sequence of amino acids) [9]. The three-dimensional structure is the folding (or conformation) of a polypeptide as a result of interactions between the side chains of amino acids that are in different regions of the primary structure.

* This work is partially supported by the Brazilian National Research Council – CNPq, under grant no. 305669/2010-9 to H.S.Lopes and CAPES-DS scholarships to C.M.V. Benítez and M.H. Scalabrin

The simplest computational model for the PFP problem is known as Hydrophobic-Polar (HP) model, both in two (2D-HP) and three (3D-HP) dimensions [5]. Although simple, the computational approach for searching a solution for the PFP using the HP models was proved to be NP-complete [3]. This fact emphasizes the necessity of using heuristic and massively parallel approaches for dealing with the problem.

In this scenery, reconfigurable computation is an interesting methodology due to the possibility of massive parallel processing. However, this methodology has been sparsely explored in molecular biology applications. For instance, [7] present a methodology for the design of a system based on reconfigurable hardware applied to the protein folding problem, where different strategies are devised to achieve a significant reduction of the search space of possible foldings. Also, [12] presents a methodology for the design of a reconfigurable computing system applied to the protein folding problem using Molecular Dynamics (MD). [13] propose a complete fine-grained parallel hardware implementation on FPGA to accelerate the GOR-IV package for 2D protein structure prediction. [4] present a FPGA based approach for accelerating string set matching for Bioinformatics research. A survey of FPGAs for acceleration of high performance computing and their application to computational Molecular Biology is presented by [11].

The main focus of this work is to develop approaches for accelerating protein folding simulations using the Harmony Search algorithm and the 3D-HP-SC (three dimensional Hydrophobic-Polar Side-Chain) model of proteins.

2 The 3D-HP Side-Chain Model (3D-HP-SC)

The HP model divides the 20 proteinogenic amino acids into only two classes, according to their affinity to water: Hydrophilic (or Polar) and Hydrophobic. When a protein is folded into its native conformation, the hydrophobic amino acids tend to group themselves in the inner part of the protein, in such a way to get protected from the solvent by the polar amino acids that are preferably positioned outwards. Hence, a hydrophobic core is usually formed, especially in globular proteins. In this model, the conformation of a protein (that is, a folding) is represented in a lattice, usually square (for the 2D-HP) or cubic (for the 3D-HP). Both 2D-HP and 3D-HP models have been frequently explored in the recent literature [9].

Since the expressiveness of the HP models is very poor from the biological point of view, a further improvement of the model is to include a bead that represents the side-chain (SC) of the amino acids [8]. Therefore, a protein is modeled by a backbone (common to any amino acid) and a side-chain, either Hydrophobic (H) or Polar (P). The side-chain is responsible for the main chemical and physical properties of specific amino acids.

The energy of a conformation is an inverse function of the number of adjacent amino acids in the structure which are non-adjacent in the sequence. To compute the energy of a conformation, the HP model considers that the interactions between hydrophobic amino acids represent the most important contribution for

the energy of the protein. Li et al. [8] proposed an equation that considers only three types of interactions (not making difference between types of side-chains). In this work we use a more realistic approach, proposed by [2], to compute the energy of a folding, observing all possible types of interactions, as shown in Equation 1.

$$H = \left(\epsilon_{HH} \cdot \sum_{i=1,j>i}^{n} \delta_{r_{ij}^{HH}}\right) + \left(\epsilon_{BB} \cdot \sum_{i=1,j>i+1}^{n} \delta_{r_{ij}^{BB}}\right) + \left(\epsilon_{BH} \cdot \sum_{i=1,j\neq i}^{n} \delta_{r_{ij}^{BH}}\right) + \left(\epsilon_{BP} \cdot \sum_{i=1,j\neq i}^{n} \delta_{r_{ij}^{BP}}\right) + \left(\epsilon_{HP} \cdot \sum_{i=1,j>i}^{n} \delta_{r_{ij}^{HP}}\right) + \left(\epsilon_{PP} \cdot \sum_{i=1,j>i}^{n} (\delta_{r_{ij}^{PP}})\right) \quad (1)$$

In this equation, ϵ_{HH}, ϵ_{BB}, ϵ_{BH}, ϵ_{BP}, ϵ_{HP}, ϵ_{PP} are the weights of the energy for each type of interaction, respectively: hydrophobic side-chains (HH), backbone-backbone (BB), backbone-hydrophobic side-chain (BH), backbone-polar side-chain (PH), hydrophobic-polar side-chains (HP), and polar side-chains (PP). In a chain of n amino acids, the distance (in the three-dimensional space) between the i^{th} and j^{th} amino acid interacting with each other is represented by r_{ij}^{**}. For the sake of simplification, in this work we used unity distance between amino acids ($r_{ij}^{**} = 1$). Therefore, δ is an operator that returns 1 when the distance between the i^{th} and j^{th} elements (either backbone or side-chain) for each type of interaction is the unity, or 0 otherwise. We also used an optimized set of weights for each type of interaction, defined by [2].

During the folding process, interactions between amino acids take place and the energy of the conformation tends to decrease. Conversely, the conformation tends to converge to its native state, in accordance with the Anfinsen's thermodynamic hypothesis [1]. In this work we consider the symmetric of H such that PFP is understood as a maximization problem.

3 Harmony Search Algorithm

The Harmony Search (HS) meta-heuristic is inspired by musician skills of composition, memorization and improvisation. Musicians use their skills to pursuit a perfect composition with a perfect harmony. Similarly, the HS algorithm use its search strategies to pursuit for the optimum solution to an optimization problem.

The pseudo-code of the HS algorithm is presented in Algorithm 1 [6]. The Harmony Search (HS) algorithm starts with a Harmony Memory of size HMS, where each memory position is occupied by a harmony of size N (musicians). Each improvisation step of a new harmony is generated from the harmonies already present in the harmony memory. If the new harmony generated is better than the worst harmony in the harmony memory, it is replaced with the new. The steps to improvise and update the harmony memory are repeated until the maximum number of improvisations (MI) is achieved.

The HS algorithm can be described by five main steps, detailed below [6][1]:

[1] For more information see the HS repository: http://www.hydroteq.com

Algorithm 1 Pseudo-code of the Harmony Search algorithm.

```
1: Parameters: HMS, HMCR, PAR, MI, FW
2: Start
3: Objective Function f(x), x = [x_1, x_2, ..., x_N]
4: Initialize Harmony Memory x^i, i = 1, 2, ..., HMS
5: Evaluate each Harmony in HM: f(x^i)
6: cycle ← 1
7: while cycle < MI do
8:    for j ← 1 to N do
9:       if random ≤ HMCR then {Rate of Memory Consideration}
10:         x'_j ← x^i_j, with i ∈ [1, HMS] {chosen randomly}
11:         if random ≤ PAR then {Pitch Adjusting Rate}
12:            x'_j ← x'_j ± r × FW {with r random}
13:         end if
14:      else {Random Selection}
15:         Generate x'_j randomly
16:      end if
17:   end for
18:   Evaluate new harmony generated: f(x')
19:   if f(x') is better than worst harmony in HM then
20:      Update Harmony Memory
21:   end if
22:   cycle ← cycle + 1
23: end while
24: Results and views
25: End
```

1. **Initialization and Setting Algorithm Parameters**: In the first step, as in any optimization problem, the problem is defined as an objective function to be optimized (line 3), which can or cannot be constrained. Originally, Harmony Search was designed for solving minimization problems [6]. The four main parameters of the algorithm are also defined here: Harmony Memory size – HMS, the Harmony Memory Consideration Rate – $HMCR$, the Pitch Adjusting Rate – PAR, and the Maximum number of Improvisations – MI.
2. **Harmony Memory Initialization**: The second step is the initialization of the Harmony Memory (HM) with a number of harmonies randomly generated (line 4). The Harmony Memory is the vector in which the best harmonies found during execution are stored. Each harmony is a vector representing a possible solution to the problem.
3. **Improvise a New Harmony**: In the third step, a new harmony is improvised based on a combination of several other harmonies found in HM (between lines 8–17). For each variable of the new harmony, a harmony of HM is arbitrarily selected by checking the corresponding probability of using or not this value ($HMCR$). If another harmony is used, the value of this variable will have small adjustments (Fret Width – FW) according to a probability (PAR). If the value of another harmony is not used, a random value

within the range of allowed values is assigned. Thus, the parameters $HMCR$ and PAR are responsible for establishing a balance between exploration and exploitation in the search space.

4. **Update Harmony Memory**: In the fourth step, each new improvised harmony is checked to see if it is better than the worst harmony from HM (lines 19–21). If so, the new harmony replaces the worst one in HM.
5. **Verification of the Stopping Criterion**: In the fifth step, the end of each iteration is checked to discover if the best harmony meets the stopping criterion, usually a maximum number of improvisations (MI). If so, the execution is completed. Otherwise, it returns to the second step until reaching the stopping criterion.

4 Methodology

This section describes in detail the implementation of the Harmony Search algorithm for the PFP using the 3D-HP-SC model of proteins. Four versions were developed: a desktop computer version and three different FPGA-based implementations. The FPGA-based versions were developed in VHDL (Very High Speed Integrated Circuit Hardware Description Language) and implemented in a FPGA (Field Programmable Gate Array) device. Two of these versions also used an embedded processor (Altera's NIOS II), as part of its hardware design. On the other hand, software implementations (i.e. for both NIOS II and the desktop computer) were developed in ANSI-C programming language.

The first hardware-based approach is a version for the 32-bit NIOS II embedded-processor, and simply reproduces the software implemented on the desktop computer. The second hardware-based approach is a version for NIOS II with a dedicated hardware block, specifically developed for computing the fitness function, as shown in Figure 1). The HS algorithm runs on the NIOS II processor and the block called "Fitness Calculation System" works as a slave of the NIOS II. The processor is responsible for initializing the Harmony Memory, improvising new harmonies, updating the Harmony Memory and, finally, distributing the individuals (also called as harmonies) to the slave block. The slave, in turn, is responsible for computing the fitness function for each individual received. The internal structure of this block is described later.

The third hardware-based approach is fully implemented in hardware and does not use an embedded processor, as shown in Figure 2. The block called "Harmony Search Core" performs the HS algorithm. The Harmony Memory initialization is performed producing a new harmony for each position of the Harmony Memory. Each variable of each new harmony is independent of the others. Therefore, each new harmony is generated in one clock pulse using a set of N random number generators, where N is the number of variables in the harmony.

Once the Harmony Memory is loaded with the initial harmonies, the iterative process of optimization of the HS algorithm is started. At each iteration, four individuals (harmonies) are evaluated simultaneously (in parallel), thus expecting an improvement in performance. In the improvisation step of the algorithm,

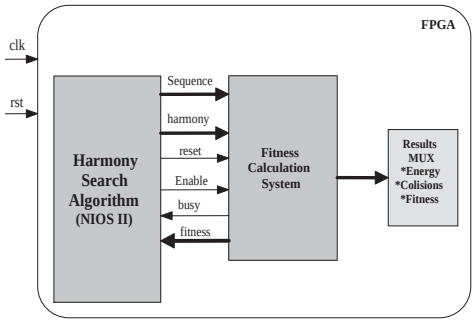

Fig. 1. Functional block diagram of the folding system with NIOS II embedded-processor.

the process of selection of each variable of the new harmony is performed independently. This procedure is done in only N clock pulses, as before.

After that, the updating of the Harmony Memory is performed by inserting the new harmonies in their proper positions. The following positions are shifted, discarding the four worst harmonies. To find the insertion position, the position of the worst harmony in the Harmony Memory is always maintained in a latch. Each variable to be replaced is treated simultaneously. Once the optimization process is completed, the best harmony found is transferred from the Harmony Memory to the "Fitness Calculation System" block in order to display all relevant information about the conformation represented by this harmony.

The chronometer block measures the total elapsed processing time of the system. The multiplex block selects output data among the obtained results (energy of each interaction, number of collisions, fitness and the processing time to be shown in a display interface).

The random number generator is implemented using the Maximum Length Sequence (MLS) pseudo-random number approach. MLS is an n-stage linear shift-register that can generate binary periodical sequences of maximal period length of $L = 2^n - 1$. In this work, we used n=7 or 4 for all probability values mentioned in the Algorithm 1, and $n = 5$ for generate variables of the new harmonies in the improvisation process.

Figure 3 shows a functional block diagram of the "Fitness Calculation System" that has three main elements: a three-dimensional conformation decoder, a coordinates memory and a fitness computation block. By calculating the energy of each different type of interactions and the number of collisions between the elements (side-chains and backbone), the fitness of the conformation is obtained. The blocks that perform such operations are described as follows:

Harmony Representation: The encoding of the candidate solutions (harmonies of the HS algorithm) is an important issue and must be carefully implemented. Encoding can have a strong influence not only in the size of the search space, but also in the hardness of the problem, due to the establishment of unpredictable cross-influence between the musicians of a harmony. There are

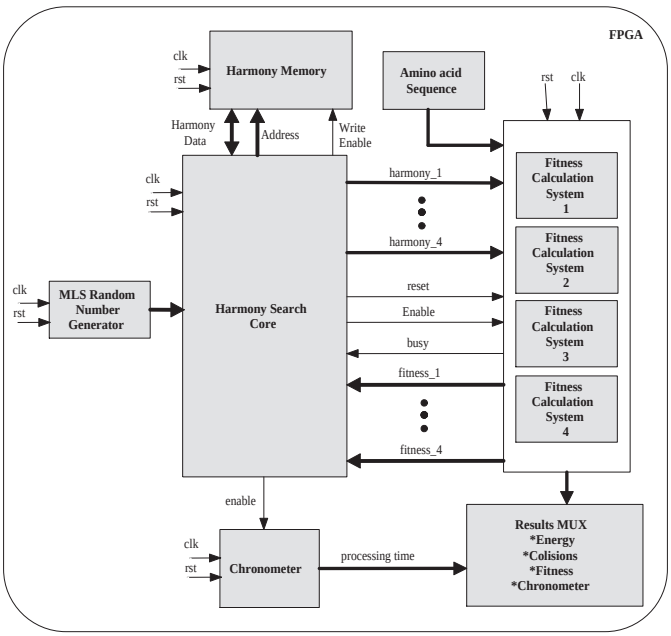

Fig. 2. Functional blocks of the proposed folding system without NIOS II embedded-processor.

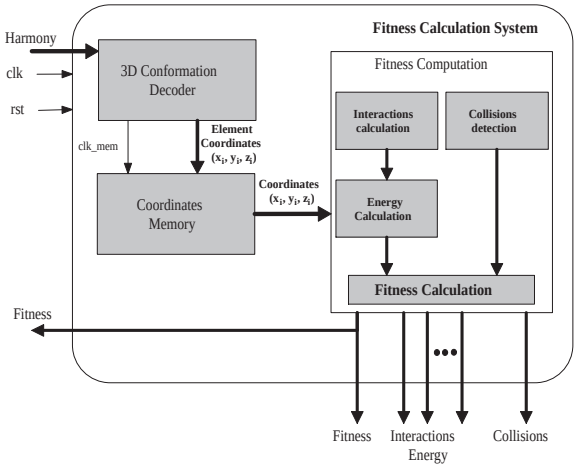

Fig. 3. Fitness computing system

several ways of representing a folding in an individual, as pointed by [9]: distance matrix, Cartesian coordinates (absolute coordinates), or relative internal coordinates. In this work we used the relative internal coordinates, because it is the most efficient for the PFP using lattice models of proteins. In this coordinates system, a given conformation of the protein is represented as a set of movements into a three-dimensional cubic lattice, where the position of each amino acid of the chain is described relatively to its predecessor.

As mentioned in Section 2, using the 3D-HP-SC model, each amino acid of the protein is represented by a backbone (BB) and a side-chain, either hydrophobic (H) or polar (P). Using the relative internal coordinates in the three-dimensional space, there are five possible relative movements for the backbone (Left, Front, Right, Down and Up), and other five for each side-chain (left, front, right, down, up). It is important to know that the side-chain movement is relative to the backbone. The combination of these possible movements gives 25 possibilities. Each possible movement is represented by a symbol which, in turn, is represented using a 5-bit binary format (number of bits needed to represent the alphabet of 25 possible movements, between 0 and 24). The invalid values ($value \geq 25$) are replaced by the largest possible ($value = 24$). Considering a folding of a n-amino acids long protein, a harmony of $n-1$ musicians will represent the set of movements of the backbone and side-chain of a protein in the three-dimensional lattice. For a n-amino acids long protein, the resulting search space is $25^{(n-1)}$ possible foldings/conformations.

Three-Dimensional Conformations Decoder: The harmony, representing a given conformation, has to be converted into Cartesian coordinates that embeds the conformation in the cubic lattice. Therefore, a progressive sequential procedure is necessary, starting from the first amino acid. The coordinates are generated by a combinational circuit for the whole conformation. These coordinates are stored in the "Coordinates Memory" which, in turn, provides the coordinates of all elements (backbone and side-chains) in a parallel output bus.

The algorithm for the decoding process (harmony $\rightarrow$ conformation) is as follows. The harmony is read and decoded into a vector using the set of possible movements. In the next step, the elements of the first amino acid are placed in the three-dimensional space. For each movement, four steps are done. First, the direction of the movement is obtained from the next movement and the direction of the movement of the predecessor amino acid. The backbone coordinates are obtained similarly from predecessor amino acid. The next step consists in determining the coordinates of the side-chain of the amino acids from the movement and coordinates of the backbone. Finally, the coordinates obtained in this process are stored in the "Coordinates Memory".

Figure 4(left) shows a conformation for a hypothetical 4-amino acids long protein, where the Cartesian coordinates of each element are represented as x_i (row), y_i (column), z_i (depth), and obtained from the relative movement of the current amino acid and position of its predecessor. Blue balls represent the polar residues and the red ones, the hydrophobic residues. The backbone and the

connections between elements are shown in gray. The search space for the protein represented in this figure has $25^{(n-1)} = 25^3 = 15625$ possible conformations. Here, the folding is formed by three movements: Ul→Dl→Dl. In this figure, the backbone and the side-chain of the first amino acid of the chain are also indicated, where the backbone and the side-chain are set to the origin of the coordinates system (0,0,0) and (0, -1, 0), respectively.

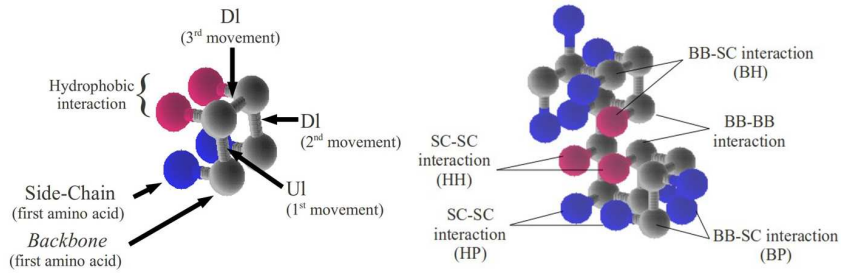

Fig. 4. Left: Example of relative 3D movements of a folding. Right: Diagram representing the possible iterations between the elements of a protein chain.

Fitness Function: In this work, we used a simplified fitness function based on that formerly proposed by [2]. Basically, this function has two terms: $fitness = H - (NC \cdot PenaltyValue)$. The first is relative to the free-energy of the folding (H, see Equation 1) and the second is a penalty term that decreases the fitness value according to the number of collisions in the lattice. The term *Energy* takes into account the number of hydrophobic bonds, hydrophilic interactions, and interactions with the backbone. Also, the number of collisions (considered as penalties) and the penalty weight are considered in this term. This penalty is composed by the number of points in the three-dimensional lattice that is occupied by more than one element (NC - number of collisions), multiplied by the penalty weight ($PenaltyValue$).

The blocks named "Interactions calculation", "Collisions detection" and "Energy calculation", compute the energy of each type of interaction (see Figure 4(right) for a visual representation), the number of collisions between elements and the free-energy (H), respectively. Finally, the block called "Fitness Calculation" computes the fitness function. It is important to note that, in the current version of the system, due to hardware limitations, all energies are computed using a sequential procedure, comparing the coordinates of all elements of the protein. As the length of sequences increase, the demand for hardware resources will increase accordingly.

5 Experiments and Results

All hardware experiments done in this work were run in a NIOS II Development kit with an Altera Stratix II EP2S60F672C5ES FPGA device, using a 50MHz

Table 1. Comparative performance of the several approaches.

n	$t_p(s)$			
	t_{pNIOS}	$t_{pNIOS-HW}$	t_{pSW}	t_{pHW}
20	557.3	54.0	6.5	1.6
27	912.8	75.0	7.7	3.0
31	1186.8	87.3	7.9	4.0
36	1460.5	107.7	9.4	5.0
48	2414.9	174.8	13.44	10.0

internal clock. The experiments done for the software version were run in a desktop computer with a Intel processor Core2Quad at 2.8GHz, running Linux.

In the experiments reported below, the following synthetic sequences were used [2], with 20, 27, 31, 36 and 48 amino acids, respectively: $(HP)^2PH^2PHP^2HP$ $H^2P(PH)^2$; $H^3P^2H^4P^3(HP)^2PH^2P^2HP^3H^2$; $(HHP)^3H(HHHHHPP)^2H^7$; PH $(PPH)^{11}P$; $HPH^2P^2H^4PH^3P^2H^2P^2HPH^3(PH)^2HP^2$ $H^2P^3HP^8H^2$.

In this work, no specific procedure was used for adjust the running parameters of the HS algorithm. Factorial experiments and self-adjusting parameters [10] of algorithms are frequently used in the literature, but these issues fall outside the focus of the work. Instead, we used the default parameters suggested in the literature. The running parameters used in this work are: $MI = 100000$, $HMS = 20$, $PAR = 30\%$, $FW = 5$ and $HMCR = 90\%$.

It is important to recall that the main objective of this work is to decrease the processing time of protein folding simulations by using the 3D-HP-SC model. Each developed approach was applied to the sequences mentioned before. Results are shown in Table 1. In this table, the first column identifies the sequence length; columns t_{pNIOS}, $t_{pNIOS-HW}$, t_{pSW} and t_{pHW} show the processing time for each approach. Where, t_{pNIOS}, $t_{pNIOS-HW}$, t_{pSW} and t_{pHW} represent, respectively, the total elapsed processing time for the NIOS II, NIOS II with the "Fitness Calculation System" block, the software and the hardware-based system without embedded processor approach. Overall, the processing time, for any approach, is a function of the length of the sequence, possibly growing exponentially as the number of amino acids of the sequence increases. This fact, by itself, strongly suggests the need for highly parallel approaches for dealing with the PFP. In order to facilitate the comparison of performance between the approaches, Figure 5 presents the speedups obtained, where:

- $S_{p_a} = t_{pNIOS-HW}/t_{pNIOS}$: speedup of the NIOS II with the "Fitness Calculation System" block relative to the NIOS II approach;
- $S_{p_b} = t_{pSW}/t_{pNIOS-HW}$: speedup of the software relative to the NIOS II with the "Fitness Calculation System" block;
- $S_{p_c} = t_{pNIOS-HW}/t_{pHW}$: speedup of the hardware-based system without embedded processor approach relative to the NIOS II with the "Fitness Calculation System" block;
- $S_{p_d} = t_{pSW}/t_{pHW}$: speedup of the hardware-based system without embedded processor approach relative to the software for desktop computers.

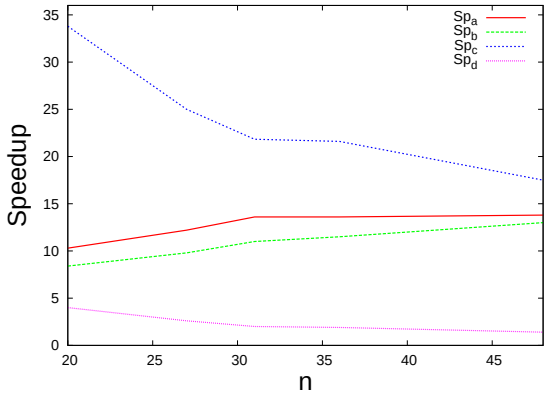

Fig. 5. Comparison of speedups between the approaches.

The NIOS II version presented the worst performance (i.e. the highest processing time) amongst all implementations. Its processing time was larger than the software approach due to the slow frequency of the internal clock (comparing with the desktop processor). It is also observed that the NIOS II with the "Fitness Calculation System" block achieved significant speedup when compared to the NIOS II approach, ranging from 10x to 13x, depending on the length of the sequence, mainly due to the number of clock cycles needed to execute each instruction in the NIOS II processor.

The hardware-based system without the embedded processor showed the best performance, mainly due to the several levels of parallelism, namely, in the Harmony Memory initialization, in the improvisation and in the parallelization of several fitness function evaluations. It is observed that this approach was significantly better when compared to the remaining hardware-based approaches, achieving a speed-up ranging from 17x to 34x, also depending on the length of the sequence. When compared with the software approach, it is observed that this approach achieved speedups ranging from 1.5x to 4.1x. The speedup decreases as the length of the sequences grows, due to the sequential procedure used to compute the energy for each type of interaction (as mentioned in Section 4).

6 Conclusions and Future Works

The PFP is still an open problem for which there is no closed computational solution. As mentioned before, even the simplest discrete model for the PFP requires an NP-complete algorithm, thus justifying the use of metaheuristic methods and parallel computing. While most works used both 2D and 3D-HP models, the 3D-HP-SC is still poorly explored (see [2]), although being a more expressive model, from the biological point of view.

Improvements will be done in future versions with the hardware-based system without the embedded processor approach, such as the full parallelization of

the energy computation. Also, future works will investigate hardware versions of other evolutionary computation approaches, such the Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) or the traditional Genetic Algorithm (GA) applied to the PFP, so as to develop parallel hybrid versions and different parallel topologies. Regarding the growth of hardware resources usage, future work will consider the use of larger devices or multi-FPGA boards.

Overall, results lead to interesting insights and suggest the continuity of the work. We believe that the use of reconfigurable computing for the PFP using the 3D-HP-SC model is very promising for this area of research.

References

1. C.B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181(96):223–230, 1973.
2. C.M.V. Benítez and H.S. Lopes. Hierarchical parallel genetic algorithm applied to the three-dimensional HP side-chain protein folding problem. In *Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics*, pages 2669–2676, 2010.
3. B. Berger and F.T. Leighton. Protein folding in the hydrophobic-hydrophilic HP model is NP-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.
4. Y.S. Dandass, S.C. Burgess, M. Lawrence, and S.M. Bridges. Accelerating string set matching in FPGA hardware for bioinformatics research. *BMC Bioinformatics*, 9(197), 2008.
5. K.A. Dill, S. Bromberg, K. Yue, and K.M. Fiebig et al. Principles of protein folding - a perspective from simple exact models. *Protein Science*, 4(4):561–602, 1995.
6. Z.W. Geem, J.-H. Kim, and G. V. Loganathan. A new heuristic optimization algorithm: Harmony search. *Simulation*, 76(2):60–68, 2001.
7. N.B. Armstrong Junior, H.S. Lopes, and C.R.E. Lima. Preliminary steps towards protein folding prediction using reconfigurable computing. In *Proc. 3rd Int. Conf. on Reconfigurable Computing and FPGAs*, pages 92–98, 2006.
8. M.S. Li, D.K. Klimov, and D. Thirumalai. Folding in lattice models with side chains. *Computer Physics Communications*, 147(1):625–628, 2002.
9. H.S. Lopes. Evolutionary algorithms for the protein folding problem: A review and current trends. In *Computational Intelligence in Biomedicine and Bioinformatics*, volume I, pages 297–315. Springer-Verlag, Heidelberg, 2008.
10. M.H. Maruo, H.S. Lopes, and M.R.B Delgado. Self-adapting evolutionary parameters: encoding aspects for combinatorial optimization problems. *Lecture Notes in Computer Science*, 3448:154–165, 2005.
11. T. Ramdas and G. Egan. A survey of FPGAs for acceleration of high performance computing and their application to computational molecular biology. In *Proc. of the IEEE TENCON*, pages 1–6, 2005.
12. W-T. Sung. Efficiency enhancement of protein folding for complete molecular simulation via hardware computing. In *Proc. 9th IEEE Int. Conf. on Bioinformatics and Bioengineering*, pages 307–312, 2009.
13. F. Xia, Y. Dou, G. Lei, and Y. Tan. FPGA accelerator for protein secondary structure prediction based on the GOR algorithm. *BMC Bioinformatics*, 12:S5, 2011.