PROSIDIS: a Special Purpose Processor for PROtein SImilarity DIScovery

A.Marongiu[1], P.Palazzari[2], V.Rosato[2]
[1]IPITEC, Rome
[2]ENEA, Computing and Modeling Unit, Rome

Abstract

This work presents the architecture of PROSIDIS, a special purpose processor designed to search for the occurrence of substrings similar to a given 'template string' within a proteome.

The paper recalls the basis of the PHG tool, developed in the framework of the HADES project, which automatically designs a parallel hardware starting from recurrence equations. In this work we present a special purpose processor, designed by PHG, which faces the protein similarity discovery problem.

Some results are given, reporting the time spent by several microprocessors and by the PROSIDIS processor to solve the same protein analysis problem.

Keywords *biological sequence analysis, FPGA, systolic architecture, System of Affine Recurrence Equations*.

1. Introduction

According to the trends in electronic system design, which foresee the use of high level synthesis methodologies and of FPGA technology to have short developing times, in the HADES project (HArdware DEsign for Scientific applications) we developed a tool to support the fast design and prototyping of dedicated co-processors to be implemented on FPGA. The tool, called PHG (Parallel Hardware Generator), is able to perform the high level synthesis of iterative algorithms starting from high level specifications and produces the VHDL code which describes the implementation of the hardware device. Such VHDL code is then processed by standard CAD tools and implemented on the FPGA.

Because of their programmability, FPGA are clocked at frequency significantly smaller than the clock frequency of general purpose processors. For this reason exploiting as much as possible the algorithm parallelism is the only way to make a FPGA-based implementation more effective than the processor based implementation of the same algorithm. For this reason PHG has been designed to produce architectures which are composed by several cooperating computing units implementing the starting iterative algorithm.

In this work we report the results achieved in the implementation of a specialized hardware device for "proteomic computation": the PROSIDIS device (PROtein SImilarity DIScovery), designed to compute the best matching region between a short amino-acid sequence and a whole proteome (section 3). Searching for the best matching between two biological sequences is one of the most used operations for protein analysis [4], as usually biologists try to infer the shape (i.e. the folding) and the function of a new protein by shape and functionalities of similar proteins. Computations involved in protein similarity analysis are based on the sum of the 'similarity' between each pair of amino-acids, thus requiring the accumulation of similarity data represented with very few bits. Furthermore, searching for the similarity between two proteomes requires $O(n^2)$ similarity comparisons, being n the length of both the proteomes. PROSIDIS can be used as HW booster for protein analysis algorithms as its operations are not efficiently supported by conventional processors which are deployed with very low efficiency in limited precision arithmetic.

In order to give a short overview about similar projects on configurable computing, we recall 1) the Cameron Project (http://www.cs.colostate.edu/cameron) in which a framework to automatically compile C programs onto programmable devices was developed, 2) the NAPA architecture [1] with the NAPA C compiler [2] which allows the partitioning of applications between fixed instruction processors and configurable coprocessors, 3) the DEFACTO project (see www.isi.edu/asd/defacto/) in which, starting from C or MATLAB specifications and using the SUIF compiler (http://suif.stanford.edu/) to extract the parallelism from the applications, the application is mapped onto a target Hardware Description Language which will be compiled onto a programmable device. A detailed review on configurable computing can be found in [3].

2. The PHG package

The Parallel Hardware Generator (PHG) package has been developed in the framework of the HADES project (HArdware DEsign for Scientific applications). The PHG theoretical framework is described in [5,6] while the detailed theory is reported in [7]. The PHG is based on the design flow shown in figure 1.

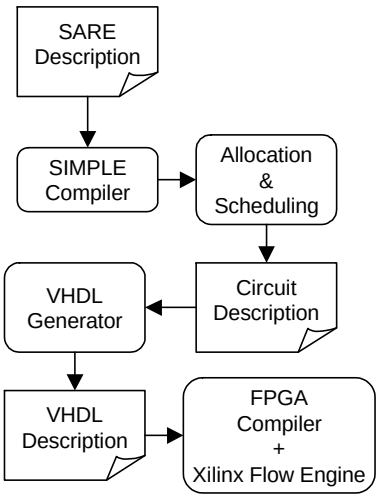

Figure 1. Layout of the design flow.

Starting from high level specifications of the problem, PHG produces a synthesizable VHDL [8]. High level specifications are given by means of a System of Affine Recurrence Equations (SARE) [9,10]. The SARE is specified through the SIMPLE (Sare IMPLEmentation) language (details on the SIMPLE language can be found in [7,11]).

In order to achieve the final circuit description, PHG performs the following steps (figure 1):

- parsing of the SARE describing the algorithm through the SIMPLE compiler and generation of the intermediate format;
- automatic extraction of parallelism by allocating and scheduling the computations through a processor-time mapping [6,7]. The mapping is represented by an integer unimodular matrix derived through an optimization process [12]. This step produces the architecture of the system expressed as a set of interconnected functional units (data path) managed by a control Finite State Machine (FSM) (data path controller) which enforces the scheduling;
- generation of the synthesizable VHDL representing the architecture determined in the previous step.
- The VHDL code is then synthesized through the standard Electronic Design Automation (EDA) tools. We used the Synopsys FPGA compiler II to produce the optimized netlist and the Xilinx Foundation Express 4 to place and route it into the target FPGA.

The theoretical foundations of the PHG derive from the works on systolic arrays conducted by many researcher in the last years [13-16]. Such works are mainly based on the projection of a regular computation domain onto a time-processor space in order to exploit the algorithm parallelism. The resulting architecture, the so-called systolic arrays, are synchronous circuits composed by a set of locally interconnected functional units. In the context of the HADES project, many efforts have been done to improve such methodologies. First of all, we considered the class of System of Affine Recurrence Equations (SARE) algorithms in order to widen the applicability domain of the synthesis methodology. In fact, using SARE, we override the local interconnectivity constraint allowing the generation of synchronous architectures with local and not local interconnections (e.g. data buses) [5,6].

An efficient optimization strategy to solve the problem of finding nearly optimal allocation and scheduling has been developed: the method is based on the simulated annealing algorithm and on a mathematical framework which allows to strongly reduce the number of candidate scheduling/allocation solutions [17].

Finally, due to the elimination of the local interconnection constraint, a new methodology has been formulated [5] in order to minimize the resources needed to implement not local communications.

The collection of the previous experiences and of the new improvements developed in the HADES project produced a visual CAD tool, the PHG, which enables the development of dedicated parallel hardware devices within a few days.

3. The PROSIDIS problem

The PROSIDIS (PROtein SImilarity DIScovery) faces the problem of finding the "degree of similarity" between a short sequence s of amino acids having length m, called peptide, and a long sequence p of amino acids having length n, representing a proteome. Each amino acid constituting the sequences can be represented as a character belonging to the alphabet

A = {I,F,V,L,W,M,A,G,C,Y,P,T,S,H,E,D,Q,N,K,R}

The typical length of p is $n = 106 \div 107$ characters. Given a peptide of length m, $s \in Am$, the PROSIDIS problem can be stated as the computation of the *n-m* values

$$M(i) = \sum_{j=0}^{m-1} DM\left(p(i+j), s(j)\right) \quad i = 0,1,...,n\text{-}m\text{-}1 \quad (1)$$

which measures the 'similarity' between the peptide *s* and a segment of p. DM(a,b) is a weighting matrix which, for any pair of amino acids a and b, returns an integer number representing the degree of similarity between them; we used the Blosum62 matrix [18]. During the sum

operation, whenever the partial value of M(i) becomes smaller than 0, it is set to 0.

A pseudo code to solve the problem is given in fig. 2.

```
input
   Proteome p(i) i=0,1,...,n-1
   Peptide s(i) i=0,1,...,m-1
   Weigthing matrix DM(i,j)
output
   Similarity M(i) i=0,1,...,n-n-1
begin
   for i = 0 to n-m-1
      M(i) = 0
      for j=0 to m-1
         M(i) = M(i) + DM(p(i+j),s(j))
         if M(i) < 0 then
            M(i) = 0
         endif
      endfor
   endfor
end
```

Figure 2: pseudo-code for the PROSIDIS Problem

Let us consider the data widths needed to correctly implement the PROSIDIS problem. Peptide and proteome sequence characters belong to the alphabet A which is constituted by 20 characters; hence each amino acid in p and s can be represented with a word length of $\lceil \log 220 \rceil$ = 5 bits. We suppose the values of the weighting matrix DM to be in the range [–8,7] and, hence, they can be represented through 4 bits in the two complement notation (weighting matrices having values outside the range [–8,7] can be appropriately scaled down). Assuming the maximum value of m = 32, the maximum allowed value for similarity measure M is 224 which can be represented with 8 bits (remember that no negative values are allowed for M).

4. Rationale to develop the PROSIDIS architecture

Let us introduce some notations in order to analyze if and when it is possible to obtain performance improvement adopting dedicated parallel devices instead of conventional COTS processors.

A program P is constituted by a partially ordered set of operations P={op_i, i=1,2,...,N} and, due to data dependencies among operations, requires a set of data transfers T={IOi, i=1,2,...,M}; t(op_i) is the reciprocal of the number of operations op_i executed per second by the system at the maximal efficiency, h_i is the efficiency with which op_i is implemented in a certain section of P (it will depend on pipeline status, data dependencies,...) and DS(IOi) denotes the data size of the IOi data expressed in bits.

A single-processor computing system is characterized by the tuple (f_{ck},R_{ck}, R_{ck}, R_{ck}), where f_{ck} is the system main clock frequency, R_{ck} is the factor to obtain the main memory clock (the system accesses main memory at the frequency $\frac{f_{ck}}{R_{ck}}$), R_{ck} is the data bus width (expressed in bits) and R_{ck} is the system Peak Speed (expressed in operations per second). As we are interested in investigating processor influence on performances, we do not take into account caches and memory size. In this paragraph we always refer to burst memory accesses, so in the evaluations of communication times the startup time for memory access is neglected; this assumption is justified by the class of problems we are referring to (iterative problems often originates sequential memory accesses).

The actual execution time T_{exe} of P on the system is within the range

$$\max(T_{comp},T_{comm}) \le T_{exe} \le T_{comp}+T_{comm}, \qquad (2)$$

being

$$T_{comp} = \sum_{op_i \in P} \eta_i \cdot t(op_i),\ T_{comm} = \frac{R_{ck} M}{f_{ck}}$$

(we are assuming that $DS(IO_i) \le W$ i=1,2,...,M, so one memory access is needed to perform one I/O operation). The left term of (2) corresponds to perfect overlapping between communication and computation phases, the right side term corresponds to the complete serialization of computing and communication phases. Both the terms are minimized when the system resources are used at their maximum efficiency, i.e. when *PS* operations per second are executed and when $\frac{f_{ck}}{R_{ck}} W$ bits per seconds are transferred. In such a case we have

$$\max\left(\frac{N}{PS}, \frac{M \cdot R_{ck}}{f_{ck}}\right) \le \text{Texe} \le \frac{N}{PS} + \frac{M \cdot R_{ck}}{f_{ck}} \qquad (3)$$

As we see from expression (3), data transfer time is likely to become the actual bottleneck in system utilization, resulting $PS >> \frac{f_{ck}}{R_{ck}}$ for the current microelectronic technology. We may refer to system granularity G_S, defined as the ratio between the processor peak speed and the maximal memory bandwidth (i.e. $G_S = \frac{R_{ck} PS}{f_{ck}}$), to have a measure of the relative influence of the communication time on the computing time: the largest is G_S, the more the communications could be the bottleneck. More precisely, starting from expression (3) we derive that communications become a bottleneck whenever G_S is larger than the program granularity G_P, defined as the average number of words to

be transferred for each operation of P, i.e. $G_P = \frac{W \times M}{\sum_{IO_i \in T} DS(IO_i)}$.

In order to override inefficiency of commercial processors, due both to the mismatch between program and system granularity and/or to the mismatch between the type of operations required by the program and the ones implemented by the system, we may design Dedicated Parallel Systems (DPS) to be implemented as Systems on a Chip. Such systems will be characterized by the right balance between computational power and processor-to-memory bandwidth: according to the number of gates available to implement the system and on the number of I/O pins for the memory interface, the system will be designed to have the largest computing power compatible with the available I/O bandwidth. Furthermore, such dedicated systems will implement in HW the operations of the algorithm, avoiding any mismatch between algorithm and system instructions.

In order to introduce a performance comparison between DPS and a single-processor computing system, let us model a DPS through the tuple (f_{ck}, R_{ck}, W, NU), where f_{ck} is the clock frequency controlling the system, R_{ck} is the factor to obtain the main memory clock (usually R_{ck} = 1), W is the global I/O width (expressed in bits) and in general it encloses as many buses as they are necessary to feed the internal circuits, NU is the number of the computing units (potentially different) which form the system data path. The system peak speed is given by $PS=NU \times f_{ck}$ -in the case that each computing unit executes one operation.

We start analyzing the computing behavior of the two systems. Referring to the conventional single-processor system, the sustained computational speed is given by $PS^{CPU} \times h^{CPU}$, being h^{CPU} the efficiency with which the processor is used. The computational speed sustained by the dedicated parallel system is given by $NU^{DPS} \times f_{ck}^{DPS} \times h^{DPS}$. The ratio between the computational speeds sustained by the dedicated parallel system and by the single processor system is

$$R = \frac{NU^{DPS} \times f_{ck}^{DPS} \times h^{DPS}}{PS^{CPU} \times h^{CPU}}. \qquad (4)$$

In order to have a significant performance improvement from the adoption of the dedicated parallel system, $R>>1$ must result. h^{DPS} is typically larger or equal to h^{CPU} because the dedicated system is designed to efficiently implement only the needed operations (often we have $h^{DPS} \approx 1$ while h^{CPU} is seldom greater than 0.5). Being the clock frequency of FPGA significantly smaller than the clock frequency of commercial processors, the key issue to have performance improvements is to exploit as much as possible the parallelism of the algorithm, i.e. it must result

$$NU^{DPS} >> \frac{PS^{CPU} \times \eta^{CPU}}{f_{ck}^{DPS}} \qquad (5)$$

Previous relation explains why the Protein Similarity Discovery problem is very likely to obtain performance improvements from FPGA implementation: in fact, being the parallel implementation based on the repetition of the simple (and small) building block executing the statement $\texttt{M(i)}_8\texttt{=M(i)}_8\texttt{+DM(p(i+j),s(j))}_4$, NU^{DPS} can be significantly large, thus satisfying relation (5) (the subscript n in notation X_n indicates the number of bits used to represent X).

In order to evaluate the right-hand term of expression (5), we need to fix a reasonable value for both the $PS^{CPU} \times h^{CPU}$ term and for the operative clock frequency of the FPGA configured with the DPS we are going to design. Taking as reference processor the ALPHA EV6.7 clocked at 667 MHz, we have PS^{CPU} =1.3; furthermore we assume an efficiency in the sequential code implementation h^{CPU} =0.5 – surely over extimated. As we are targeting the Xilinx XV1000-4 FPGA, a reasonable value for the clock frequency of a complex design is f_{ck}^{DPS} =25MHz. Substituting values in the right-hand side of expression (5) we obtain the condition $NU^{DPS} >> \frac{1.3 \times 10^9}{2 \times 25 \times 10^6} \approx 27$ which states that, in order to obtain a performance improvement - with respect to an ALPHA EV6.7 processor - through the implementation of a DPS on an FPGA dedicated to solve the Protein Similarity Discovery problem, we must be able to design and implement a number of parallel basic functional units (FU) significantly larger than 27. As each FU performs the summation between two numbers (expressed respectively with 4 and 8 bits) and implements a LUT with 400 entries and with the output expressed by 4 bits, it seems very reasonable to assume that a large number of FU's could be implemented on the same FPGA.

Referring to the communications, the single processor system has theoretical peak bandwidth given by

$$BW^{CPU} = \frac{f_{ck}^{CPU}}{R_{ck}^{CPU}} \times W^{CPU} \qquad (6)$$

Similarly, the Dedicated Parallel System has the theoretical peak bandwidth expressed by

$$BW^{DPS} = \frac{f_{ck}^{DPS}}{R_{ck}^{DPS}} \times W^{DPS} \qquad (7)$$

Both the CPU and the DPS use their bandwidth with efficiency $\eta_{BW}^{D} = \frac{< L^{D} >}{W^{D}}$, being $D=\{CPU,DPS\}$ and $<L^{D}>$ the mean value of the width of data (expressed in bits) transferred by D. It is worth to underline that usually $\eta_{BW}^{DPS} = 1$, because W^{DPS} is designed to match the data width, while it may result $\eta_{BW}^{CPU} < 1$ whenever $<L^{CPU}>$ is smaller than W^{CPU}: this happens when data put in parallel on the bus cannot be processed in parallel by the CPU. For instance, on a 32 bit system, it is possible to check in parallel if 6 pairs of 5 bit strings are equal; it is sufficient to code the i-th string (i=0,…,5) on the data bits ranging from 5(i+1)-1 down to 5i and to perform the ex-or between the two words containing the 6 strings; on the contrary, if we want to add 6 pairs of 5 bit numbers represented in the 2-complement notation, we cannot simply add the two words containing the 6 pairs of numbers (there are problems with the negative numbers), so we have to read the data pairs sequentially. In the first case we have $<L^{CPU}>$=30 bit while in the second case $<L^{CPU}>$=5 bit.

From previous reasoning, the ratio between the sustained CPU and DSP bandwidth (assuming R_{ck}^{DPS}=1) is given by

$$R_{BW} = \frac{R_{ck}^{CPU} \times f_{ck}^{DPS} \times W^{DPS}}{f_{ck}^{CPU} \times < L^{CPU} >} \qquad (8)$$

In order to have communication performance improvements when implementing the algorithm on a dedicated parallel system, R_{BW}>1 must result. This happens when

$$W^{DPS} > \frac{\left(f_{ck}^{CPU} / R_{ck}^{CPU} \right)}{f_{ck}^{DPS}} \times < L^{CPU} > . \qquad (9)$$

Also in this case the parallelism is the key issue to hide the effects due to the disadvantageous ratio between clock frequency of FPGA devices and commercial processors.

Let us now evaluate expression (9). From the analysis of the algorithm in figure 2, we know that, for each operation, two 5 bit characters must be loaded and one 8 bit number must be stored. In the case of a sequential implementation on a CPU such an I/O traffic results in 18 bits transferred in 3 bus cycles, yielding $<L^{CPU}>$=6 bit. We refer again to an ALPHA based system (namely the DS20) in which the memory is accessed at 133 MHz. Assuming for the DPS implemented on the FPGA the memory access rate at 25 MHz, relation (9) becomes

$$W^{DPS} > 32$$

which means that data I/O bus width in the FPGA must be larger than 32 bits. This condition can be fulfilled by exploiting data parallelism (that is parallelism on the outer loop, being completely independent all its instances). As each instance of the outer loop reads amino acid $p(i+j)$ (i.e. 5 bits) and returns the matching score $M(i)$ (8 bits), the internal pipelined structure has W^{P}=13; it is sufficient to put three of such pipelined units in parallel to have $W^{DPS} = 39 > 32$.

As both the relations (5) and (9) are likely to be fulfilled in the case of the Protein Similarity Discovery problem, a Dedicated Parallel System implemented on FPGA technology is a good candidate to significantly improve sustained performances with respect to a system based on conventional processors.

5. PROSIDIS Design

Following the HADES design flow, computation of (1) is expressed in SARE form through the SIMPLE program shown in figure 3. The program starts with the definition of the algorithm indices and parameters, of the input variables and of the result variables. After the sections with definitions, the algorithm equations follow, each one being specified with its data dependencies (the indices of variables appearing as arguments of the function) and its validity domain (the inequalities between curl brackets). In particular, equation 1 initializes the M values to 0, equation 2 propagates the M values along the j direction and, finally, equation 3 performs the accumulation of the amino acid similarity. Last statement defines the output of the algorithm. A SIMPLE algorithm, representing a collection of SARE, has a computing domain which can be represented in the Cartesian space. For the SIMPLE algorithm represented in figure 3, the computing domains are shown in figure 4.

Going on with the PHG design flow as reported in figure 1, the optimization process, used to determine allocation and scheduling, discovered the projection matrix $\begin{pmatrix} \Lambda \\ \Sigma \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ which allocates each element of the peptide to a different functional unit; in fact, applying the projection matrix to the computing domain, we obtain $\begin{pmatrix} t \\ p \end{pmatrix} = \begin{pmatrix} \Lambda \\ \Sigma \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$ which originates the equality $p=j$ (j=-1,…,m-1) (according to [5,6], t is the temporal coordinate and p the processor coordinate in the transformed (time-processor) space). The used projection matrix originates a linear pipeline structure because dependence relation from equation 3, namely from `M[i,j-1]` to `M[i,j]` and represented by the dependence vector $\begin{pmatrix} d_i \\ d_j \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, is projected into the transformed dependence vector $\begin{pmatrix} d_t \\ d_p \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$,

i.e. data produced at time t by processor p will be used at time (t+1) (d_t=1) by processor (p+1) (d_p=1).

Finally, the VHDL source, which contains nearly 5000 lines of code, was automatically produced.

```
Ind [i,j];
Par[n,m] {m>=1,n>=1};

/*String sequence*/
Input p[1] {0<=i<=n-1};

/*Peptide sequence*/
Input s[1] {0<=i<=m-1};

/*Output sequence*/
Result M;

/*Equation 1*/
M[]=InitMatch();
{-m+1<=i<=n-1,j=-1};

/*Equation 2*/
M[] = PropagateMatch();
{i+j<=-1,j>=0,i>=-m+1};

/*Equation 3*/
M[]=AddMatch(p[i+j],s[j],M[i,j-1]);
{0<=i+j<=n-1,0<=j<=m-1};

/*Output*/
Write(M[]);
{-m+1<=i<=n-m,j=m-1};
```

Figure 3: SIMPLE program expressing the SARE to solve the Protein Similarity Discovery Problem

6. PROSIDIS Architecture and Test Bed Configuration

We designed the PROSIDIS architecture specialized for the case of a proteome with length n=2,096,000 and a peptide of length m = 24. The basic pipelined architecture, automatically produced by the PHG package according to the projection matrix described in previous paragraph, is shown in figure 5. It is composed by a data path (the pipelined structure) driven by the data path controller which enforces the algorithm scheduling. Small boxes represent delay elements (edge triggered registers) while the big boxes represent the computing units. The Blosum62 weighting matrix (appropriately scaled down) has been implemented through a Look Up Table (LUT). Each computing element performs the comparison of two amino acids by means of the LUT. The LUT output is then added to the output of the previous stage and the result is transmitted to the next stage unless it is a negative value; in such a case 0 is transmitted. We underline that the VHDL corresponding to the combinatorial circuit in the square box in figure 5, implementing the elementary function behavior, is the only code written directly by us.

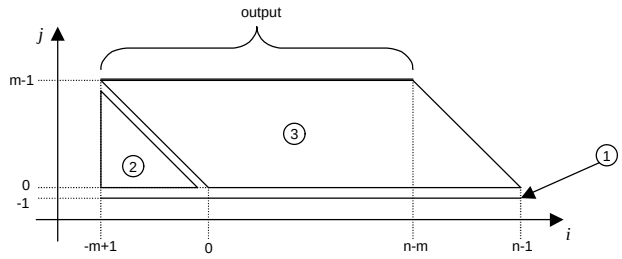

Figure 4: computing domain for the SARE in figure 3

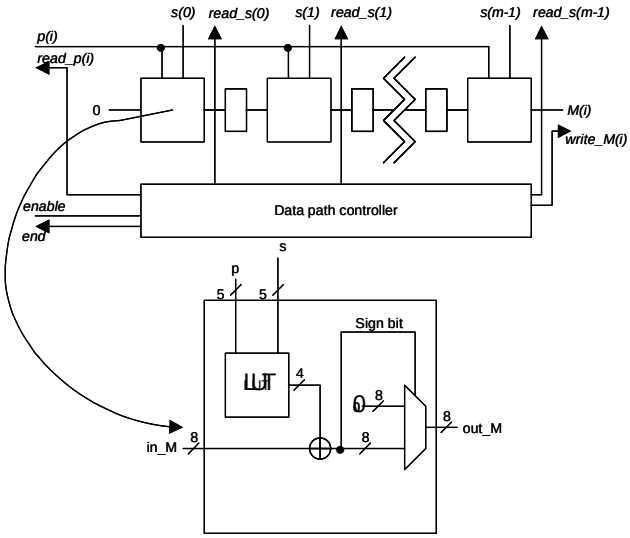

Figure 5: basic pipelined architecture to solve the Protein Similarity Discovery problem

The test bed we used to implement the PROSIDIS architecture is a prototyping board equipped with a PCI interface (33 MHz), 8 MB of SRAM memory and one Xilinx Virtex XV1000 FPGA. The board is hosted by a standard PIII@550 MHz PC.

Due to the large size of the FPGA device and in order to increase the computing power of the PROSIDIS architecture, we replicated 4 times in the same FPGA the pipelined architecture of figure 5. The resulting FPGA structure is depicted in figure 6 where it is shown the additional circuitry needed to interface the pipelined structures to the board devices.

The PROSIDIS architecture is able to perform the contemporaneous comparison of 4 proteome sections of length n=524000 against a single peptide of length m=24. The clock cycles needed to carry out the whole computation are m+n=524024. Constrained at the speed grade of the FPGA we used (XV1000-4), the synthesized

design is clocked at a frequency $f_{clk} = 30$ MHz, corresponding at a sustained computing rate of 2.88×10^9 operations per second and to a sustained bus I/O bandwidth of 1.56 Gb/s.

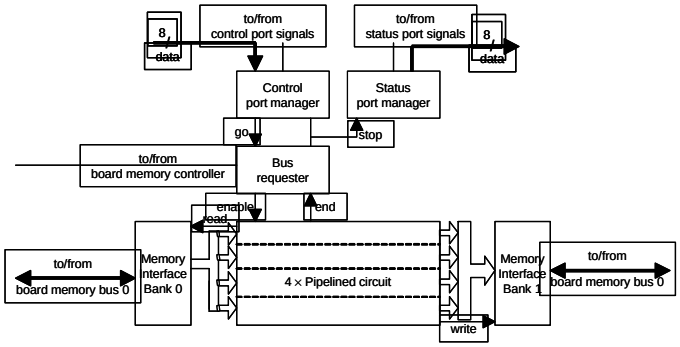

Figure 6: Block diagram of the PROSIDIS architecture

Referring to the Protein Similarity Discovery problem, the core of the computation is demanded to the PROSIDIS processor while the global control flow (loading/reading data to/from board memory, starting the FPGA computation) is demanded to the PIII host computer. The pseudo code is sketched in figure 7 and is constituted by a sequence of calls to functions of the prototyping board:

- DMA from the host memory to the board memory of the peptide sequence;
- DMA from the host memory to the board memory of the 4 proteome sequences *p*. Notice that, being each proteome character represented with 5 bits, the 4 proteomes can be transferred within a single DMA operation because they are byte-aligned in a 32 bit word;
- Write the start value on the control port; this command triggers the start of FPGA computation;
- Wait for FPGA completion through the reading of the End value from the FPGA status port;
- DMA from the board memory to the host memory of the 4 similarity sequences. Notice that, being each similarity value represented with 8 bits, the 4 sequences can be transferred within a single DMA operation because they are byte-aligned in a 32 bit word

7. Results

In order to test the advantages attainable by using the PROSIDIS dedicated processor as booster for a conventional sequential system, we implemented the algorithms to solve the Protein Similarity Discovery problem on different general-purpose platforms. Particular effort has been spent to optimize the SW implementation of the algorithm on the test systems. The test processors have been chosen among those available in our research center.

```
input
   4 Proteome p(i) i=0,1,...,523999
   Peptide s(i) i=0,1,...,23
output
   4 Similarity M(i) i=0,1,...,523975
begin
   DMA(host2board,s)
   DMA(host2board,p)
   WRITE_CONTROL_PORT(Start)
   READ_STATUS_PORT(End)
   DMA(board2host,M)
end
```

Figure 7: Pseudo-code of the program to solve the Protein Similarity Discovery problem

Results have been compared with those obtained on the test bed architecture consisting of a 550 MHz Pentium III processor connected to the RC1000-PP prototyping board equipped with a Xilinx XV1000 FPGA.

The architectures used in the tests are:

- 1000 MHz Pentium III, 32 bit, 512K of L2 cache, Win2000, MS Visual Studio C++ V6.0;
- 667 MHz Alpha EV6.7 (API UP2000 board), 64 bit, 4M of L2 cache, Linux Kernel 2.3.14, Compaq C compiler (ccc) V6.2-506;
- 450 MHz Sun UltraSparc II, 64 bit, 4M of L2 cache, Solaris 2.7, Sun WorkShop C compiler V5.0;
- 200 MHz IBM Power3, 64 bit, 4M of L2 cache, AIX 4 OS, C for AIX compiler V4.4.
- 300 MHz R12000 SGI Onyx 2, 8M of L2 cache, MIPS C compiler.

Table 2 summarizes the results reporting, for each algorithm and machine configuration, the time spent to execute the searching for a sequence with m=24 amino acids on a proteome with n=2,096,000 amino acids.

We underline that the PIII+FPGA implementation takes into account both the overhead to manage the sequential parts of the applications and the times necessary to start the FPGA and to DMA data to and from host memory: 35 ms, out of 52 ms, are devoted to DMA operations. This fact implies that the simple technological upgrade from the 32 PCI clocked at 33 MHz to the 64 PCI clocked at 66 MHz would reduce the DMA overhead to ~9 ms. The other technological improvement, passing from the XV1000 to the XV2000 FPGA, will allow the implementation of 8 parallel pipelines instead of 4 (the prototyping board has still two unused memory banks); this fact would reduce the PROSIDIS computing time to 8.5 ms, giving an overall computing time of ~17.5 ms. So the implementation of PROSIDIS with a fastest PCI and the last generation XV2000 FPGA would triple the speedup figures reported in Table 2. It is worth to note that PROSIDIS shows speedup factors ranging from 5.6

up to 55.6, thus supporting with experimental data the theoretical analysis about the advantages derivable from FPGA implementation of the Protein Similarity Discovery problem.

	Computing time (msec)	FPGA Speed up
PIII@550+FPGA	52	1.00
PIII@1000	290	5.6
EV6.7@667	387	7.4
SGI R12000@300	533	10.3
Power3@200	1152	22.2
UltraSparc@450	2892	55.6

Table 2: Elapsed times (milliseconds) to solve the Protein Similarity Discovery problem on different platforms. The last column shows the speed-up values achieved by the FPGA-based implementation with respect to the specific platform.

8. Conclusions

The work presented the design and implementation of a special purpose processor, called PROSIDIS, devoted to solve the Protein Similarity Discovery problem arising in the field of protein analysis. The design has been carried out through the Parallel Hardware Generator (PHG) tool developed by two of the authors in the framework of the HADES project (HArdware DEsign for Scientific applications). After a brief review on PHG, the Protein Similarity Discovery problem has been introduced and a detailed analysis has been performed in order to explain why it is possible to obtain significant performance improvements with respect to commodity off the shelf processors (COTS). The design, based on the theory on recurrence equations and their projection, has been illustrated together with experimental results where the performances attained by COTS based systems were compared with the ones obtained by an FPGA implementation of the PROSIDIS processor on a prototyping board hosted by a Pentium III personal computer. Results evidence speedup figures ranging from 5.6 up to 55.6, clearly demonstrating the validity of the proposed design.

References

[1] C.R. Rupp, M. Landguth, T. Garverick, E. Gomersall, H. Holt, J.M. Arnold, and M. Gokhale: 'The NAPA Adaptive Processing Architecture', Proc. IEEE Symp. on FPGAs for Custom Computing Machines, 1998

[2] M.B. Gokhale and J.M. Stone: 'NAPA C: Compiling for a Hybrid RISC/FPGA Architecture', Proc. IEEE Symp. on FPGAs for Custom Computing Machines, 1998

[3] Katherine Compton, Scott Hauck: 'Configurable Computing: A Survey of Systems and Software'. Tech. Report from the Northwestern University, Dept. of ECE, 1999

[4] S.F. Altschul, W. Gish, W. Miller, E. W. Myers, D. J. Lipman, " Basic local alignment search tool", J. Mol. Biol. 215 (1990) 403.

[5] A.Marongiu, P.Palazzari: “Automatic Implementation of Affine Iterative Algorithms: Design Flow and Communication Synthesis”. Comp. Phys. Comm., vol.139 (2001).

[6] A.Marongiu, P.Palazzari, “Automatic Mapping of System of Affine Recurrence Equations (SARE) onto Distributed Memory Parallel Systems”, IEEE Trans on Soft. Eng., 26 (2000) 262.

[7] A.Marongiu, "Hardware and Software High Level Synthesis of Affine Iterative Algorithms", Ph.D Thesis in Electronic Engineering, University "La Sapienza"(Rome), 2000.

[8] IEEE standard VHDL language reference manual. IEEE std. 1076-1993

[9] C.Mongenet, P.Clauss, G.R.Perrin, “Geometrical Tools to Map Systems of Affine Recurrence Equations on Regular Arrays”, Acta Informatica, Vol. 31, No. 2, pp. 137-160, 1994.

[10] V.Loechner, C.Mongenet, “OPERA: A Toolbox for Loop Parallelization”, International Workshop on Software Engineering for Parallel and Distributed Systems, 1996.

[11] A.Marongiu, P.Palazzari, “High Level Software Synthesis of Affine Iterative Algorithms onto Parallel Architectures”. Proc of the 8th Int. Conf. HPCN Europe 2000, Amsterdam.

[12] A.Marongiu, P.Palazzari, “Optimization of Automatically Generated Parallel Programs”, Proc. of the 3rd IMACS International Multiconference CSCC'99, Athens.

[13] Y.K.Chen, S.Y.Kung, “A Systolic Design Methodology with Application to Full-Search Block-Matching Architectures”, J. of VLSI Signal Proc., Vol. 19, pp. 51-77, 1998.

[14] J.Bu, E.F.Deprettere, P.Dewilde, “A Design Methodology for Fixed-Size Systolic Arrays”, Proc. of Int. Conf. ASAP 90, pp. 591-602, September 1990.

[15] H.Lim, E.E.Swartzlander jr., “Efficient Systolic Arrays for FFT Algorithms”, The 29th Asilomar Conference on Signals, Systems and Computers, pp. 141-145, 1995.

[16] N.L.Passos, E.H-M.Sha, “Scheduling of Uniform Multi-Dimensional Systems under Resource Constraints”, IEEE Trans. on VLSI Systems, Vol. 6, n.4, pp. 719-730, Dec1998.

[17] G. Deodati “Optimisation for the automatic synthesis of digital circuits” (in italian). Master thesis in electronic Engineering. University “La Sapienza” - 2001.

[18] S. Henikoff, J.G. Henikoff, “Amino acid substitution matrices from protein blocks”. Proc. Natl. Acad. Sci. USA 89:10915-10519, 1992.