High Level Synthesis for the Cray XD-1 System in a Grid Environment

Giorgio Brusco *(a)*, **Rocco Casilli** *(a)*, **Philip LoCascio** *(d)*, **Alessandro Marongiu** *(a,b)*, **Silvio Migliori** *(b)*, **Paolo Palazzari** *(a,c)*, **Vittorio Rosato** *(a,c)*, **Paolo Verrecchia** *(a)* [1]

(a) Ylichron S.r.l, (b) ENEA – INFO, (c) ENEA – CAMO, (d) ORNL – Life Science Division

ABSTRACT: *The rapid increase in integrated circuit densities, with the slowing in the increase in commodity CPU clock rates, has fostered the investigation of hybrid architectures using FPGA technologies to provide some specialized accelerated computational capability, such as the Cray XD-1 system. In order to take full advantage of the FPGA enabled nodes of the Cray XD-1 system, and avoiding the traditional problematic development times typical of custom digital circuit design, High Level Synthesis (HLS) methodologies appear to be a promising research avenue. An HLS allows one to obtain a synthesizable VHDL description, in a near automated way, starting from a specific algorithm given in some High Level language (such as the ANSI C). In this presentation we describe the using of the HLS methodology developed by Ylichron S.r.l. – a spin-off company of the Italian Agency for the New Technologies, the Energy and the Environment (ENEA) – to implement a library of core algorithms utilizing the FPGA module of the XD-1 system. In order to further promote the using of such powerful computing platform, the XD1 system has been embedded within the ENEA Grid environment. As illustrative test case to show the advantages achievable through the adoption of reconfigurable devices we implemented a parallel solution, coded in C+MPI, of the Low Autocorrelation Binary String optimization problem; the solution boosted by the FPGA devices configured to compute the cost function of the optimization process was characterized by a speed-up factor S=19 with respect to the case in which FPGAs were not used.*

KEYWORDS: GRID computing, High Level Synthesis

[1] This work has been performed under the framework of the project CRESCO (Center for Computational Research on Complex Systems), funded by the Italian Ministry for University and Research, in the frame of the National Research Programme (PON 2000-2006 Measure II.2, Action a).

1. Introduction

Reconfigurable computing gained a lot of attention in last years as it was promising the achieving of performances significantly higher than the ones given by the current generation of high end computing systems. The reason for such a performance gain is based on the possibility to use reconfigurable devices (i.e. FPGA, Field Programmable Gate Array) to implement, for each algorithm to be executed, dedicated computing architectures which match the algorithm structure. As a consequence to this approach, all the FPGA available logic, as well as I/O bandwidth, will be used with maximal efficiency, thus avoiding the severe underusing of silicon area usually encountered in conventional microprocessors which have a lot of their area devoted to implement complex memory hierarchies, out of order execution, scheduling policies etc... In a conventional processor, at each clock cycle, a lot of resources do not perform any operation while, resorting to a dedicated parallel computing architecture, nearly each part is active at every clock cycle. This parallel structure of the architecture implemented on the reconfigurable device is the key issue to overcome the gap existing between the clock frequency of microprocessors and FPGA devices: such a gap is in the order of a factor 10÷30, being the clock frequency of a microprocessor ~3 GHz, while 100-300 MHz is the typical range of operating frequency for the circuits implemented onto FPGAs. As a rule of the thumb, taking into account the different efficiency in the resources utilization, an implementation on FPGA becomes competitive with a SW implementation when the parallelism exploitable is larger than the processor to FPGA clock frequency ratio.

Following such a promise for high performances, they have been proposed, by Cray and SGI, computing architectures with FPGA devices tightly connected to the processing nodes through fast switches and not through the PCI. Thanks to this tight connection, FPGA/Processor access to data produced by the Processor/FPGA is very fast (low latency and high bandwidth). Nevertheless, the availability of a fast and reliable parallel computing architecture equipped with reconfigurable devices is only a necessary condition for the exploitation, by the end users, of the computing power of the system. In order to allow the end users to take real advantage from a reconfigurable architecture like the Cray XD1 – the one which we refer to – it is necessary that the access to the FPGA part should not require skills different from the ones of normal users of computing systems. To this aim, the FPGA part will be activated by the end users through calls, within a standard C+MPI code, to a standard library of functionalities. Furthermore, it would be highly desirable that the system could be immersed in a Grid environment.

In this work we will present the experience we made on the integration of the Cray XD1 system, equipped with FPGA co-processing elements, in the ENEA Grid environment and in the developing the way to integrate the FPGA library into a C+MPI code. Some library elements have been developed through the High Level Synthesis design flow made available by Ylichron S.r.l

2. The ENEA GRID Environment

In order to meet the needs of the medium and small Italian companies, ENEA (Italian National Agency for New Technologies, Energy and the Environment) has started a Grid project. The main goals of this project are:

a) providing ENEA for its internal use, and the eventual use by external users, with a computing Grid.
b) stimulating the demand of Grid computing by medium and small companies, providing examples of successful applications running on the Grid, and helping the users in formulating their models in a way that can exploit the power of the Grid.
c) developing a special middleware to support the seamless integration of special purpose devices (i.e. electronic microscopes, DNA sequencers, FPGA and DSP, ...).

ENEA research activity is performed in 12 sites around Italy, and the ENEA Grid infrastructure is born with the mission to provide an integrated environment covering most of ENEA's multi-platform computational resources, connected in a wide area network. The key elements in the ENEA Grid implementation are:

1) a distributed file system (AFS/OpenAFS),
2) a wide area resource manager (Platform LSF) and
3) a unified user interface (based on Java and Citrix Technologies).

AFS is a distributed filesystem product, pioneered at Carnegie Mellon University. It offers a client-server architecture for file sharing, providing location independence, scalability, security, and transparent migration capabilities for data.

LSF provides on demand access and scalability in distributing applications and executing workload. It is based on a multiqueue system of batch jobs, having the queues programmable service policies. The ENEA Grid implementation selected a time based queue classification; each queue can feed every system, selected later on the basis of user specified flags or requests (number of CPUs, specific SW, class of architecture (FPGA), ...)

These components form the middleware that provides the functionalities typical of a Grid, "unique authentication, authorization, resource access and resource discovery," as stated by Ian Foster and Carl Kesselman in their classical paper "The Anatomy of the

Grid." The choice of mature and multi-platform software components, either proprietary as LSF and Citrix or open-source as OpenAFS, greatly improves the reliability, ease of management and update of the system. ENEA Grid implementation began in 1998 and the new resources have been progressively introduced in what is now a production-quality Grid infrastructure.

The services offered by ENEA Grid include the access to computation resources, graphic and 3-D immersive facilities, and software resources, including commercial codes (e.g., Fluent, Abacus, Catia) and elaboration environments (e.g., Matlab, IDL, SAS). ENEA Grid computational resources include about 700 processors with either MIMD architecture -- as IBM SP, SGI Altix and Onyx, and Cray XD1 -- or in cluster configuration -- with Linux 32/64 systems and Apple Mac OSX. A 128-node IBM SP4 with 256 nodes IBM SP5 [~3 Tflops] are the most important resource at the moment. These resources are located in six computer centres in northern, central and southern Italy and are connected over the WAN by the Italian Academic and Research Network (GARR). ENEA Grid has, at present, about 600 registered users and more that 2TB of stored data, with 18TB available.

3. The ENEA High Level Synthesis design flow

As stated in the introduction, the key point for the success of reconfigurable architectures is the availability of a design flow which allows the (nearly) effortless migration toward such a new computational paradigm. The path we are following in the integration of the CRAY XD1 system within the ENEA Grid is based

1) on the adoption of a library of functionalities implemented onto FPGA and easily integrable, through few API calls, within a C+MPI program and
2) on the development of a design flow to enlarge, in a fast and reliable way, the set of functionalities included in the library.

The first point is achieved in a straightforward manner, by including within a C function the calls to the Cray primitives to load the desired bitstream into the FPGA, to perform the (eventually) needed DMA operations to move the input data and to output the results and to start the application. The functionality can be invoked in a blocking or not blocking way; in the first case the calling process will wait till the completion of the functionality, in the latter case the calling process will prosecute its execution and will call a specific synchronization function to wait, when necessary, the results produced by the FPGA.

In order to define a simple design flow to develop a new functionality to be added to the FPGA library, we decided to use High Level Synthesis (HLS) methodologies and, in particular, we used the HLS methodology developed by Ylichron S.r.l, an ENEA spin-off company. We decided for such a solution because, as illustrated in figure 1, specifics can be expressed by means of an ANSI C program (actually it is a subset of the ANSI C, as the use of pointers and recursion is disallowed). The user is not requested to deal with low level details like parallelism extraction and signal representation.

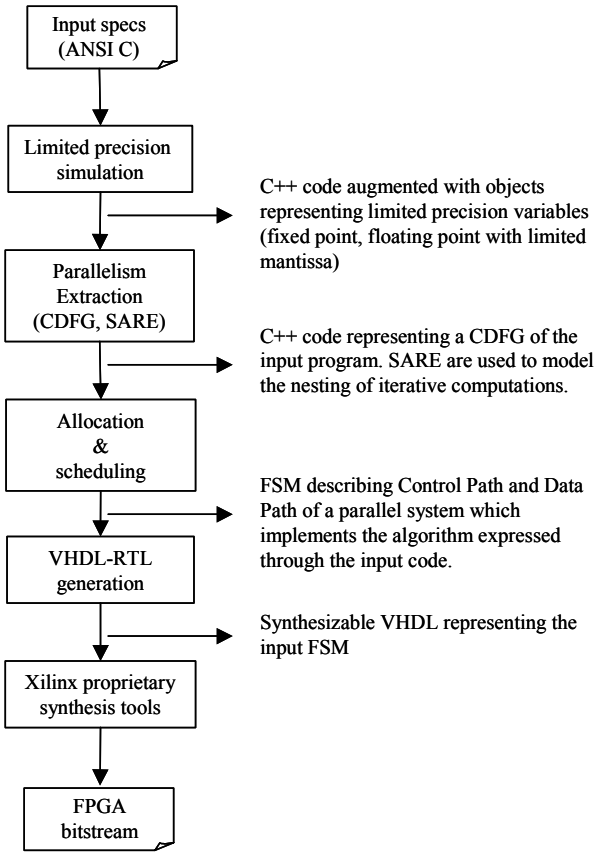

Figure 1: Ylichron S.r.l. design flow

Once that the user has given the original specifics through a C program, together with a collection of significative input cases, the code is automatically instrumented in order to estimate the dynamic of the program variables, thus making an initial guess for potential candidates for a limited precision, fixed point representation. Such a guess is given as input to a simulator which, for all the input tests, performs an error analysis with respect to the original representation; guided by the results of the error analysis, corrections are manually performed on the signal representation an the process is iterated till the satisfaction, within a certain threshold error, of the specifics. The code obtained from previous iterative process is used as input for a module in charge to extract the parallelism from the original program; control flow and data flow analysis are

performed and the result is a Control and Data Flow Graph (CDFG), with nodes characterized by a blocking semantics in order not to be obliged to use buffered communication channels in the synthesis process. The multidimensional variables, accessed within iterative constructs, are analysed through the System of Affine Recurrence Equations theory [Mong 94] [Mar 00]. The CDFG, as well the SARE portions of the code, are then passed to an Allocation and Scheduling module which uses a priority list scheduling algorithm [Lak 1999] to map CDFG onto a Finite State Machine (FSM) controlling a parallel data path; the parts described through the SARE model are mapped according to the method described in [Mar 01]. Finally there is a module to generate the synthesizable VHDL program which is successively processed through standard proprietary compilation tools to produce the bitstream which will configure the FPGA. It is worth to be underlined that, once the specifics are correct and the limited precision arithmetic maintains the errors within the predefined threshold, the final architecture – corresponding to the bitstream – will implement the specifics within the same error threshold, being the whole synthesis process 'correct-by-construction'.

In order to standardize I/O operation and to facilitate the integration of the automatically generated architectures within the Cray environment, we defined an HW interface layer which allows to access, in arbitrated manner, the memory blocks of the VirtexPro FPGA (the SelectBlockRAM), a set of predefined internal registers and a DMA engine which transfers data to/from the FPGA memory resources.

4. Example of MPI application

As a prototype example of the proposed approach, we decided to implement a parallel algorithm to determine to length N string $S_N = \{s_1, ..., s_N; s_i=\pm 1\}$ which (nearly) minimizes the sum of the square of all the autocorrelation lags, i.e.

$$S_N = \min_{S} \left(\sum_{k=1}^{N} \left(\sum_{i=0}^{N-k} s_i s_{i+k} \right)^2 \right)$$

the former is known as the Low Autocorrelation Binary Sequences (LABS) problem. [Mar 94], [Fer 00].

The LABS problem arises from the area of digital communication. In fact, LABS's can be used to generate efficient codes for error correction and robust procedures for communication synchronization. The problem of finding LABS's is usually tackled by using an efficient optimization strategy which minimizes the cost function obtained by cumulating the square of all the autocorrelation lags $H_k(s)$:

$$H(s) = \sum_{k=1}^{N} \left(\sum_{i=1}^{N-k} s_i s_{i+k} \right)^2 = \sum_{k=1}^{N} H_k^2$$

Due to its complexity, optimization heuristics are used to minimize H(s). In this work, we have chosen the Parallel Tempering method, proposed by Marinari and Parisi [Mar 92], as it has been shown to give better results, in this specific problem, than other heuristics. The pseudo-code for the PT algorithm is reported in figure 2, where the $MC(T_i)$ notation indicates the execution of a Monte Carlo algorithm at temperature T_i (see figure 3)

```
input
   M replicas of MC
   temperature values T_i for each MC
output
   s* | H(s*)<= H(s) ∀s generated
begin
   for i=1 to M s[i] = random
   while not stop_condition
      for i=1 to M do in parallel MC(Ti)
      for i=1 to M-1 do in parallel
          swap(s[i+1],s[i]) with
            p=exp((1/T_{i+1}-1/T_i)*(H(s[i+1])-H(s[i]))
         /* s[i] becomes the solution for MC at
            temperature T_{i+1} and s[i+1] becomes
            the solution for MC at temperature T_i*/
      endfor in parallel
   endwhile
end
```

Figure 2: skeleton of the Parallel Tempering algorithm

```
/* Monte Carlo cycle at temperature T, MC(T) */
for i=1 to NumberSamples
   x' = GenerateNew(r,x) /* x' is 'close' to x */
   accept = false
   if f(x') < f(x) then
      accept = true
   else if exp((f(x)-f(x'))/T) > random(0,1) then
      accept = true
   end if
   if accept then x = x'
end for
/* end of Monte Carlo cycle at temperature T, MC(T) */
```

Figure 3: skeleton of a Monte Carlo cycle at temperature T

From figure 2, we see that all the Monte Carlo (MC) cycles can run in parallel and, after the end of each MC cycle, they try – still in parallel – to swap the solutions with the adjacent (with respect to the temperature) MC. As at each step of the MC cycle the function H(s), which has time complexity $O(N^2)$, has to be computed and the other operations (generation of a new string, acceptance or rejection of the new string) are O(1) and practically, for typical N values, they require a negligible computing time with respect to the computation of H(s), in our analysis and optimization effort we will refer only to the computation of H(s).

As typically requested when implementing an algorithm onto an FPGA, the algorithm to compute H(s) requires $O(N^2/2)$ operations (bit comparison and increment/decrement of an accumulation variable) while the overhead to move results to/from the FPGA memory space requires (N/64+1) 64 bit data movements, thus

being negligible with respect to the computation. Furthermore, H(s) computing involves serial bit level operations, forcing a really inefficient use of the processor resources. Referring to the processor-to-FPGA clock frequency ratio as stated in the introduction, due to highly inefficient use of processor resources, it results that a degree of parallelism of 30 should be enough to obtain a real advantage from an FPGA implementation of the algorithm to compute H(s). When we analyse a possible pipelined parallel implementation of the H(s) computation, we easily recognize that the H_k autocorrelation lags can be computed in parallel, thus introducing a parallelism degree of N (in our tests we adopted the value N=1024 >> 30). Using the Ylichron design flow, which in this case heavily relies on the SARE theory, we obtained the parallel architecture sketched in figure 4, which uses N Functional Units (whose structure is depicted in figure 5), one squaring unit $(\,)^2$ and one accumulation unit.

In order to compute the whole H(s) function such an architecture requires 2N clock cycles, resulting in an average efficiency in FU utilization

$$\eta = \frac{\#\text{operations}}{\#\text{FU} \times \#\text{Clock Cycles}} = \frac{\left(\frac{N^2}{2}\right)}{(N+2)\times(2N)} \approx 0.25$$

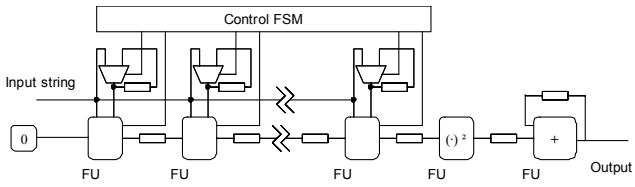

Figure 4: Structure of the pipelined architecture to compute H(s)

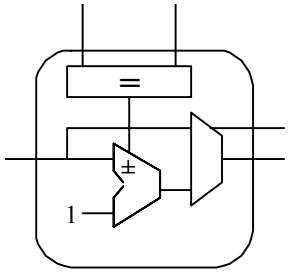

Figure 5: Structure of each one of the N FUs in the pipelined architecture shown in figure 4

Once automatically obtained the bitstream encoding previous architecture, the FPGA library function which implements the H(s) computation is organized as shown in figure 6.

As we see, the Compute_H function receives, other than the I/O parameters, an integer value which is the handle for the FPGA. Such an handle has to be initialized through a call to the load_FPGA API which, receiving the complete path to the configuration bitstream, to the FPGA driver and an integer value containing some FPGA configuration flags, configures the FPGA downloading the specified bitstream into the selected FPGA and returning the handle. Once the bitstream has been downloaded into the FPGA, the circuit is started through a call to the application_start_blocked(fp_id) API which does not return the control till the end of the FPGA computation.

The dma_transfer_blocked, as well as the read_long_word_from_FPGA are some of the API implemented referring to the HW layer designed to interface the designs produced with the FPGA Cray environment.

```
void Compute_H(u64 *s, u64 *res, int fp_id)
{
  dma_type = HOST_TO_FPGA;
  dma_size = (N/8);
  ptr_host = (u_64) s;
  resource_fgga = selramblock;
  dma_transfer_blocked(dma_type,
                       dma_size,
                        ptr_host,
                        resource_fgga,
                       fp_id);
  /* Start application and wait for its completion*/
  application_start_blocked(fp_id);
 /* Read the result from FPGA Register0 */
  resource_fgga = reg0;
  *res = read_longword_from_FPGA(resource_fgga, fp_id);
}
```

Figure 6: Skeleton of the FPGA library function which implements H(s) computation.

In order to give an idea of the impact on performances of the FPGA utilization, we report that, on a single node of the Cray XD-1 system in the case of a binary string with length N=1024, the SW implementation of the H(s) function requires 970 μsec. As reference, the main portion of the C code adopted is reported in figure 7; it is worth to be underlined that the integer product operation resulted faster than equality check followed by increment/decrement and that, in order to avoid misleading and unfair cache effects, we took care to put the string in cache before starting the computation. When we resorted to the FPGA function Compute_H (fig. 6) on strings of the same length N, thus including the I/O data transfer time, we measured that the computing time drops down to 52 μsec (having clocked the FPGA circuit at f_{ck}=130 MHz). So in the LABS problem the adoption of FPGA, programmed through the automatic design flow sketched in figure 1, results in an overall speed-up S=19.

```
for (k=0; k<SIZE; k++) {
        temp = 0;
        for (i=0; i<(SIZE-k); i++)
            temp += s[i] * s[i+k];
        autocorr += temp*temp;
    }
```

Figure 7: reference C code to compute H(s)

5. Conclusions

In this work we reported about the experience we made in ENEA using the Cray XD1 system, equipped with FPGA co-processing elements (the Xilinx VirtexPro P50). The pursued approach has been mainly concerned with the 'usability' of the system, trying to hide, as much as possible, all the low level details which are normally involved in the use of reconfigurable devices. To this aim, the High Level Synthesis design flow defined by Ylichron S.r.l., based on the acquisition of specifics through ANSI C programs and on correct-by-construction steps which transform the ANSI C description into a parallel architecture encoded in the configuration bitstream.

In order to make really easy the exploitation of programmable devices, a library of functionalities has been defined, so that the user accesses the FPGA simply calling a function which hides all the low level details (I/O transfers, synchronizations). To further enlarge the usability and availability of the XD1 system, it has been embedded within the ENEA Grid environment.

As explanatory example to illustrate the using and the advantages of programmable devices within C+MPI programs, we considered the solution of the Low Autocorrelation Binary String problem; in such a case the adoption of the proposed design flow allows to speed-up the computation of a factor S = 19.

Acknowledgments

The authors would like to thank Alessandro Secco (LSF specialist) and CRAY staff for their precious help in the set-up of the CRAY environment within the ENEA Grid. We are also in debt with Giovanni Bracco, the ENEA AFS expert.

References

[Fer 00] Ferreira F., Fontanari J., Stadler P.: "Landscape statistics of the low-autocorrelation binary string problem", J. Phys. A: Math. Gen. 33 (2000) 8635-8647

[Lak 1999] Lakshminarayana, G., Khouri, K., Jha, N.K.: "Wavesched - a novel scheduling technique for control-flow intensive Designs", IEEE Trans. on CAD of Integrated Circuit and Systems, 18, 5 (May), 1999.

[Mar 00] Marongiu A., Palazzari P.: "Automatic Mapping of Systems of N-dimensional Affine Recurrence Equations (SARE) onto Distributed Memory Parallel Sustems", IEEE Transactions on Software Engineering, vol. 26, n. 3, March 2000.

[Mar 01] Marongiu A., Palazzari P.: "Automatic implementation of affine iterative algorithms:Design flow and communication synthesis", Computer Physics Communications, 139, pp. 109-191, Sep. 2001.

[Mar 92] Marinari E., Parisi G.: "Simulated Tempering: A New Monte Carlo Scheme", Europhysics Letters, 19, 1992.

[Mar 94] Marinari E., Parisi G., Ritort F.: "Replica Field Theory for Deterministic Models: Binary Sequences with Low Autocorrelation", J. Phys. A: Math. Gen. 27 (1994) 7615.

[Mong 94] Mongenet C., Clauss P., Perrin G.R.: "Geometrical Tools to Map System of Affine Recurrence Equations on Regular Arrays", Acta Informatica, Vol. 31, No. 2, pp. 137-160, 1994.

About the Authors

Giorgio Brusco is SW development engineer at Ylichron Srl – g.brusco@ylichron.it

Rocco Casilli is SW development engineer at Ylichron Srl – r.casilli@ylichron.it

Philip LoCascio is researcher in Genome Analysis and Modeling, ORNL

Alessandro Marongiu (Ph.D.) is CTO for the HW section of Ylichron Srl and researcher at the ENEA Portici Research Centre. a.marongiu@ylichron.it

Silvio Migliori is in charge of the Grid projects at ENEA – migliori@enea.it.

Paolo Palazzari (Ph.D.) is CTO for the SW section of Ylichron Srl and researcher at the ENEA Casaccia Research Centre - p.palazzari@ylichron.it

Vittorio Rosato (Ph.D.) is president of Ylichron Srl and researcher at the ENEA Casaccia Research Centre - v.rosato@ylichron.it

Paolo Verrecchia is HW development engineer at Ylichron Srl – p.verrecchia@ylichron.it