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)*

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

Outline of presentation

- Programming with reconfigurable devices
  - Why
  - When
  - How
- High Level Synthesis methodologies
- Using FPGA in C+MPI environment
- Example: the LABS problem
- Conclusions

Programming with reconfigurable devices: Why

- Processors are hitting the “memory wall”: the gap between the processor clock period and the memory access time is increasing so
- A lot of area is devoted to implement memory hierarchies (both cache controllers and cache banks)

Programming with reconfigurable devices: Why

- In conventional processor there is poor exploitation of the concept of parallelism, being data and istructions fetched from memory;

- the instruction per cycle (IPC) figure is low, in the order of 5-6;

Programming with reconfigurable devices: Why

- Some computations (limited precision arithmetic, character based) severely under use processor resources (memory bandwidth, functional units);

- in these cases processors are employed with very poor efficiency.

Programming with reconfigurable devices: Why

- Programmable devices (FPGA) offer a lot of interesting characteristics:
  - Very High Internal Memory Bandwidth: 250 32bit SelectRam Banks, accessed at f_{ck}=200 MHz, offer a bandwidth of 200 GB/sec;
  - High Degree of Internal Parallelism: depending on the type of computation, from tens up to thousands of functional units can be implemented and activated in parallel, resulting in very high IPC

Programming with reconfigurable devices: When

FPGA can be effectively used whenever a computationally intensive problem

- involves operations not efficiently supported by COTS processors,
- under-uses processor bandwidth and/or processor functional units,
- has enough parallelism to hide the lowest clock frequency of FPGA circuits with respect to processors

Programming with reconfigurable devices: When

- includes an heavy computational kernel which can be mapped onto a parallel architecture embeddable within a programmable device,
- communication between such a computational kernel and the remaining part of the algorithm has complexity smaller than the computational kernel.

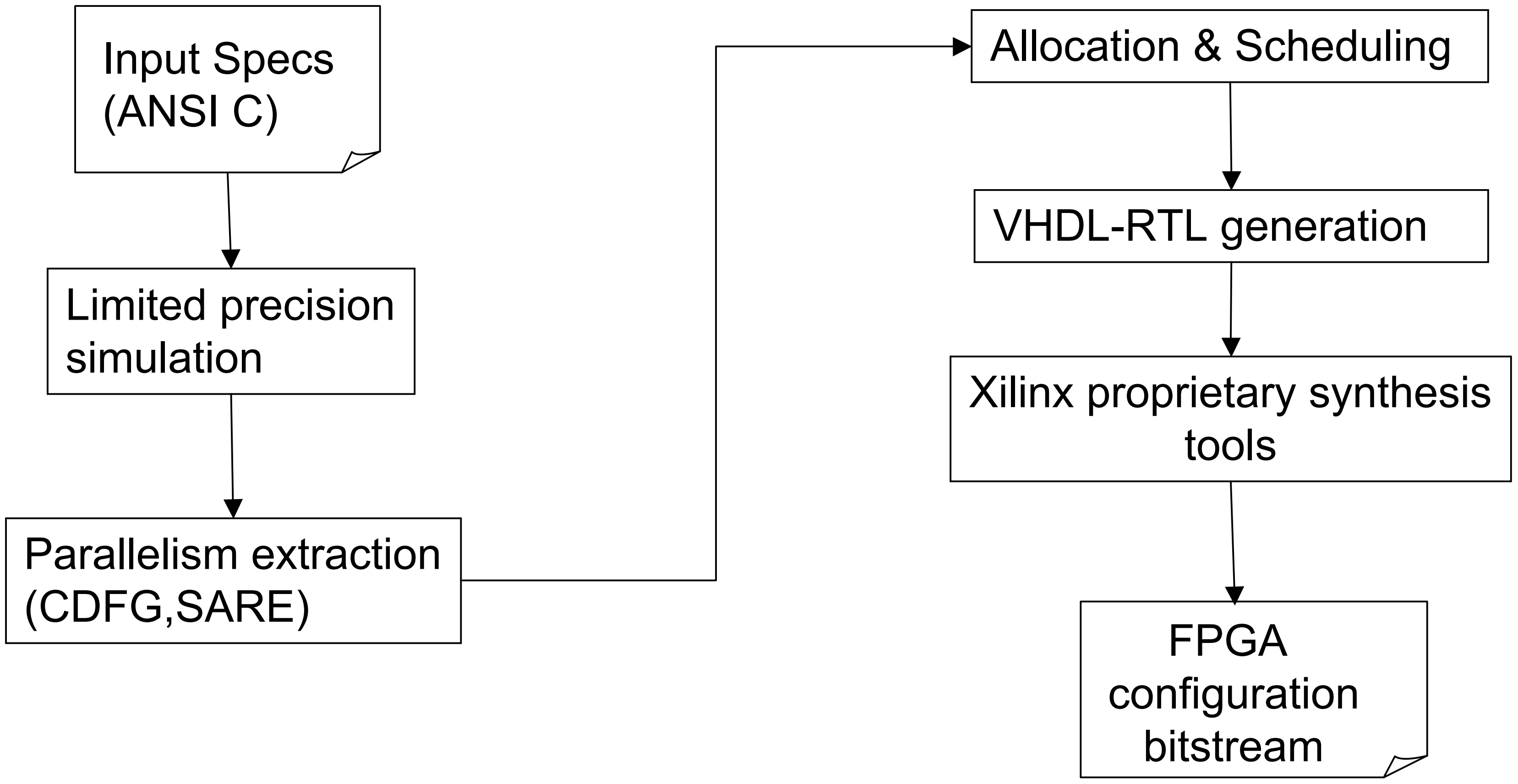
Programming with reconfigurable devices: How

- Simplicity should be a must:
- the access to the FPGA part should not require skills different from the ones of normal users of computing systems.
- High Level Synthesis methodologies have to been adopted in order to hide, as much as possible, HW low level details.
- Within a certain extent, performances can be sacrificed to obtain simplicity

High Level Synthesis methodologies

- HLS translates, in a sequence of (nearly) automated steps, HL specifics (expressed, for instance, in ANSI C) into a synthesizable description (VHDL-RTL, for instance).
- The whole translation process has to be correct by construction, so the final architecture is correct once the correctness of the original C specs has been ensured
- Specs are checked at HL

The Ylichron Design Flow


Input Specs (ANSI C)
Limited precision simulation
Parallelism extraction (CDFG,SARE)
Allocation & Scheduling
VHDL-RTL generation
Xilinx proprietary synthesis tools
FPGA configuration bitstream

Using FPGA in C+MPI

- The structure of a C library of FPGA functionalities has been defined.
- A given functionality is invoked as a normal C library function; this function embodies all the data transfer and synchronization details.
- We defined a standard environment, plus the necessary API, to access – in arbitrated manner – FPGA resources (memory, registers, DMA)

The LABS problem

- $S = \{s_1, \ldots, s_N;\ s_i = \pm 1\}$

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

- 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 LABS problem: Parallel Tempering

```
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
```

The LABS problem: Monte Carlo

```
/* 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) */
```

The LABS problem

- Is a good candidate for FPGA implementation?
- involves operations not efficiently supported by COTS processors?
- Yes, single binary operations have to be implemented

The LABS problem

- Is a good candidate for FPGA implementation?
- under-uses processor bandwidth and/or processor functional units?
- Yes, each operation requires the access at only two different bits, so we use only 3% of memory bandwidth. Furthermore, we use only one integer unit.

The LABS problem

- Is a good candidate for FPGA implementation?
- has enough parallelism to hide the lowest clock frequency of FPGA circuits with respect to processors?
- Yes, it has nearly N functional units that can run in parallel, and

  $N=\mathbf{0}(1000) >> f_{ck\_proc}/f_{ck\_circuit} = 3x10^9/1x10^8=30$

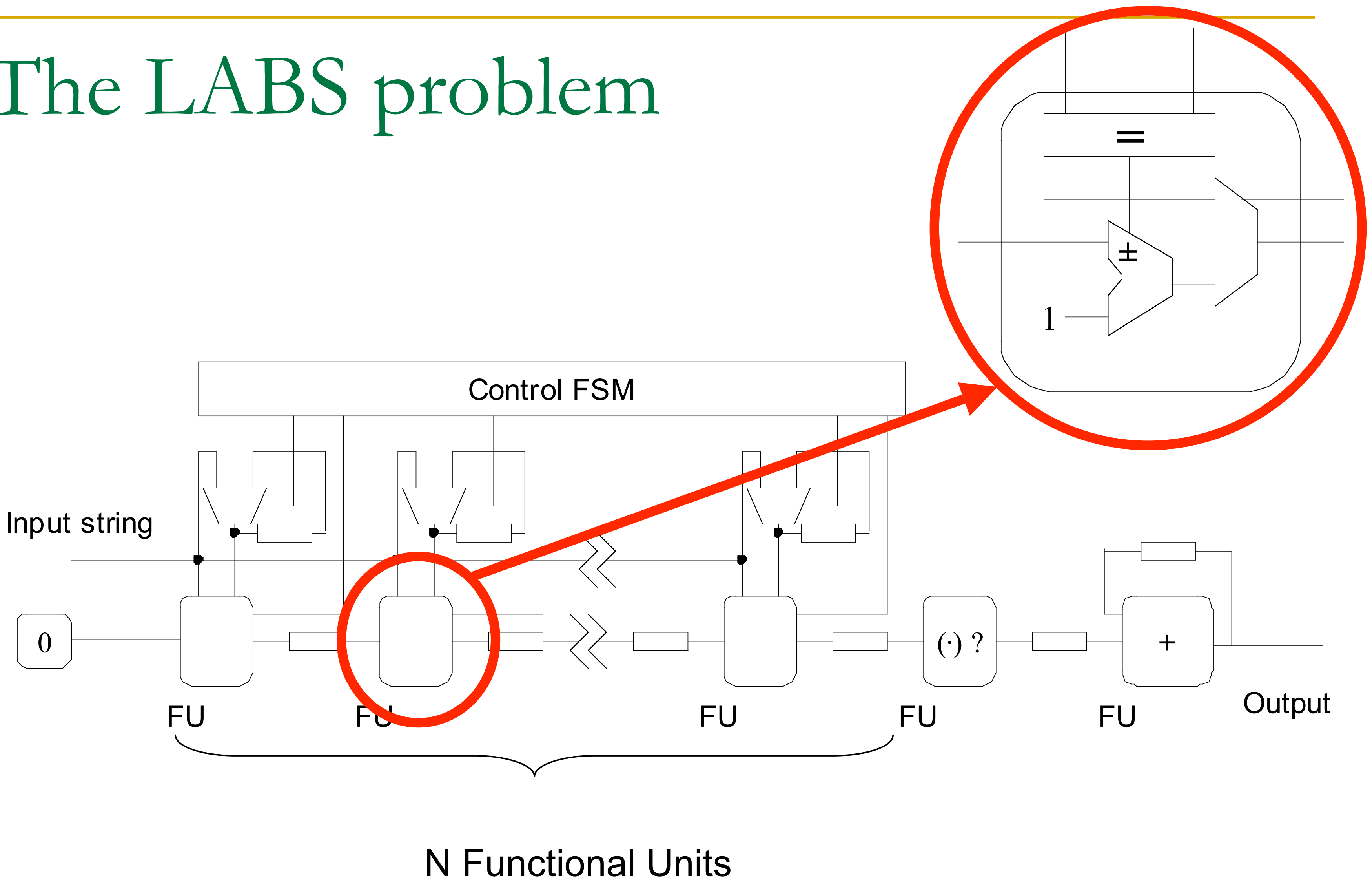
The LABS problem

- Is a good candidate for FPGA implementation?
- includes an heavy computational kernel which can be mapped onto a parallel architecture embeddable within a programmable device?
- Yes, it is based on a computational kernel which has $O(N^2)$ time complexity; a lot (thousands) of bit operations can be mapped into a single FPGA.

The LABS problem

- Is a good candidate for FPGA implementation?
- communication between such a computational kernel and the remaining part of the algorithm has complexity smaller than the computational kernel?
- Yes, communication is $O(N)$ while computation is $O(N^2)$

The LABS problem



The LABS problem

```
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);
}
```

Program DMA

Do DMA

Do computation

Pass results back

The LABS problem: performances

- N = 1024;
- Optimized implementation on the Cray XD-1 Opteron processor: 970 μsec to compute the H(s) function.
- Implementation on FPGA + communication and synchronization overhead: 52 μsec;

  circuit clocked at 100 MHz and VirtexPro P50 used at ~80%
- S = 19

Conclusions

- We discussed why, when and how it is convenient to use FPGA devices instead of SW implementation on conventional processors;
- The Ylichron HLS design flow has been presented;
- The illustrative LABS problem has been shown as example of the performance improvements that can be obtained with HLS and dedicated circuits.