Method for run time hardware code profiling for algorithm acceleration

Vladimir Matev, Eduardo de la Torre and Teresa Riesgo
CEI - Centro de Electrónica Industrial
Departamento de Automática, Ingeniería Electrónica e Informática Industrial
Universidad Politécnica de Madrid (UPM).
C/ José Gutiérrez Abascal, 2, 28006, Madrid, Spain

ABSTRACT

In this paper we propose a method for run time profiling of applications on instruction level by analysis of loops. Instead of looking for coarse grain blocks we concentrate on fine grain but still costly blocks in terms of execution times. Most code profiling is done in software by introducing code into the application under profile witch has time overhead, while in this work data for the position of a loop, loop body, size and number of executions is stored and analysed using a small non intrusive hardware block. The paper describes the system mapping to runtime reconfigurable systems. The fine grain code detector block synthesis results and its functionality verification are also presented in the paper. To demonstrate the concept MediaBench multimedia benchmark running on the chosen development platform is used.

Keywords: System on Chip, Algorithm Acceleration, FPGAs, Profiling, Reconfigurable Systems, Partial reconfiguration.

1. INTRODUCTION

Over the last decades a lot of research effort is spend on System on Chip (SoC) design solutions in order to cover the constantly increasing requirements newly appeared application. One important challenge in the design of optimized System on Chip (SoC) is the profiling based acceleration of resource demanding algorithms.

Profiling is wide used technique and gives a good result as a first step to software or hardware algorithm acceleration. By means of code profiling of the target application system designers identify compute intensive code in order to optimize it and reduce the execution time [1]. As it is demonstrated, a great amount of computation time is spent on relatively small loops within the code and the optimization of this specific piece of code has a great impact in the overall application execution time. More recently, designers has start studying the possibility of taking advantage of inherited hardware parallelism in order to provide further optimization to repetitive and computing intensive loops and drastically reduce the execution time, hardware acceleration. Usually, software profiling is required for choosing a candidate block for future hardware implementation and the decision of the code partitioning, which part is going to be executed in hardware, is taken at design time For instance in the MOLEN approach [2], specific instruction are added to the software source code and/or to the microprocessor instruction set to switch a task execution into a specific hardware block defined upon software profiling at design time. The resulting HW/SW system is highly optimized and the achieved speedups are from several to thousands of times depending on the application domain and the system architecture. However, the resulting system is application specific.

Still software profiling is intrusive as it introduces some special code in order to measure times of execution and to create profiling information and it only can be done in compile time.

On the other hand, the approach followed in this paper is focused on hardware profiling in order to provide the basics for a runtime adaptive system. The main advantage of a hardware profiling solution is that it does not require compiler or code modifications and it is nonintrusive. There are other hardware based profiling solution, like [3], were a frequent loop cache controller is connected to an address bus and a specific additional signal for short backyard bunch is required for the profiling process Nevertheless, the provided results are based on simulation and not on real HW.

Differently the hardware profiling proposed in this paper targets real reconfigurable system. The paper proposes an approach for optimal hardware selection, based on loop execution time and provides further implementation of the loop profiler on a real reconfigurable hardware platform.

The rest of the paper is organized as follows. In the next section 2, the general idea of the runtime profiling approach presented in this paper is presented. The hardware implementation issues on an FPGA based runtime reconfigurable systems are widely discussed in section 3. The test performed and some results are provided in section 4 and finally conclusions can be found in section 5.

2. GENERAL HARDWARE PROFILING SYSTEM FOR DYNAMIC ACCELERATION

This paper targets reconfigurable systems that are basically a combination of a reconfigurable array and a microprocessor or microcontroller with different degree of coupling: form separate devices, to tightly coupled systems where both elements share the same silicon die. The acceleration solution based on hardware profiling presented in this paper targets such tightly coupled system where a microprocessor is embedded into a Filed Programmable Gate Array (FPGA). This section describes the general hardware profiling solution for software acceleration in reconfigurable systems and afterwards the complete acceleration flow is overviewed.

2.1 Hardware Profiling System for Software Acceleration

A block diagram of the complete acceleration system is shown on Figure 1 and each block is described next.

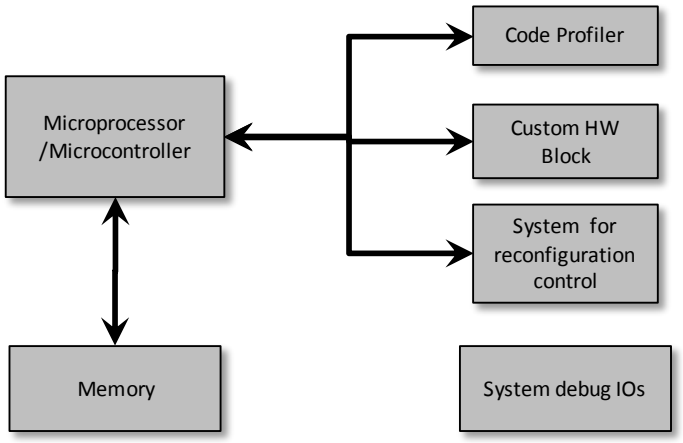

Figure 1. Block diagram of the hardware profiling based acceleration solution for reconfigurable hardware.

A main subject when dealing with profiling is the granularity. In this work, instead of looking for coarse grain blocks we concentrate on fine grain but still costly blocks in terms of execution times. By choosing a small grain solution and especially by focusing on a small but very frequently repeated code blocks good acceleration ratio is achieved maintaining a small footprint of the hardware blocks. In this approach, the profiling granularity is directly related to the hardware reconfiguration granularity, the smallest piece of hardware that will be loaded into the custom HW block (see Figure 1).

The most important piece of the system, and the main topic of this paper, is the hardware profiler block (Code Profiler in Figure 2.). The profiling block is tightly coupled with the microprocessor and decodes brunch instructions in order to detect loops during code execution. By decoding the instruction the address of the loop's beginning and size of the loop body is extracted. Additionally we use a counter for the number of iterations for every loop analyzed so every time the program we analyze passes trough the loop the value of the counter increases.

A wrong selection of the tasks to be executed in hardware could result in system performance losses due to the delay of the own hardware, that is usually ignored, and the hardware to software communication delays. Therefore, it is also very important to know not just the number of times a loop is executed, but also the execution time per iteration. By combining the loops frequency with the needed loop execution time, more complete data for the best candidate to be implemented in hardware is obtained. For that purpose in the hardware profiler, proposed in this paper, clock cycles between loop iterations are kept in a counter.

As a result, during the hardware profiling process, data for the position of a loop, loop body, size number of executions and execution time is stored and analyzed in the hardware profiling block.

The basic goal is to implement loops that can be parallelized using FPGA specific resources, such as multipliers and adders in order to achieve smaller execution times and as a consequence overall algorithm acceleration. In such system, an important trade-off is the area occupied by the hardware block, which is directly related to the needed reconfiguration time, and the clock cycles to finish the operation [3]. Cost function in order to select the most appropriate candidate includes as an additional information not only the frequency, size and body of the loop but the minimal time of loop´s iteration meaning that the best hardware loop implementation will strongly depends not only on the frequency of the loop but on the time overhead that can be achieved by parallelizing the block compared to the time in software execution.

This important future is introduced with no additional cost in terms of processor clock cycles as detailed in section tree.

Apart from the described blocks, the system also includes a memory to hold hardware configuration. Currently there is not a fast hardware synthesis solution and therefore predefined system configurations have to be held in a local configuration library that is mapped to the memory.

2.2 Dynamic Acceleration – General Workflow

At the initial state algorithm is executed on the embedded processor with the profiling block running and collecting data for a number of loops within the code and the custom hardware block is empty. On the next stage loop data is analysed by the hardware profiler block in order to find the loops that is the best candidate for being switched to hardware. The results from the analysis are retrieved to the microprocessor that in the next step, selects an appropriate hardware configuration taking into account the mentioned cost parameters and the available free area in the custom hardware block. After this, the configuration is taken from the configuration library and sent to the reconfiguration control system. Finally, special instructions are added to the program memory to pass the execution of this piece of the software to the hardware.

Currently hardware configuration is selected from a predefined library, but research efforts are being put on fast hardware synthesis. In a future phase of our work it will be done by using decompiling techniques [4] and JIT compilation [5]. Every hardware block must have a proper description in order to match with the code extracted from the loop body. Once a mach is found the pattern witch is previously synthesized and stored in an external RAM memory as a partial bit stream is to be programmed in the FPGA. This is done via partial bitstream reconfiguration described in the following section.

3. RUNTIME PROFILING SYSTEM IMPLEMENTATION

This section described the implementation of the presented in the previous section profiling approach. For this aim, first a runtime reconfigurable system have been designed for a commercial Virtex II Pro based board and is described in detail in the next section and afterwards the implementation of the profiling block in that platform is presented.

3.1 Reconfigurable System Implementation

The target reconfigurable system is a tightly coupled commercial device, a Virtex II Pro FPGA provided by Xilinx. A general view of this FPGA architecture can be seen on Figure 2 [1].

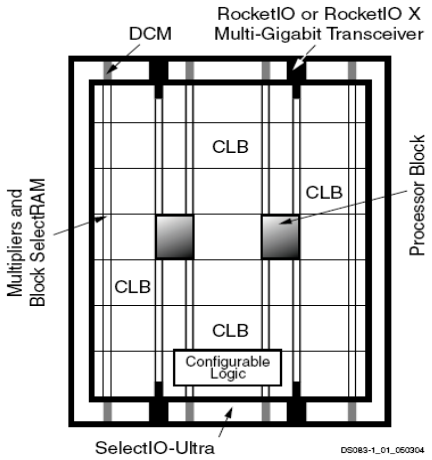

Figure 2, Virtex II Pro FPGA Architecture [1].

Virtex II Pro FPGAs as any other FPGA could be seen as a sea of reconfigurable elements called Configurable Logic Blocks (CLBs) with a set of interleaved specific resources that are meld into the silicon die. For this FPGA these resources are: embedded memories Block RAMs (BRAMs), Digital Clock Management Blocks and embedded processor blocks. The processor blocks are one or two IPB Power PC 405 32-bit RISC processors that have has a 5 stage pipeline, separate 16 kB instruction and data L1 caches and a Core Connect bus. The processor can run with speed of up to 300 MHz. The FPGA array is surrounded by Input Output Block (IOBs) that are organized into IO banks. The device provides several reconfiguration interfaces: an external serial JTAG configuration interface, an external parallel Select Map interface and an 8 bit parallel Internal Configuration Access Port (ICAP) that permits to reconfigure the FPGA from the inside, being the own FPGA system the responsible of the reconfiguration process. Regarding the reconfiguration methods, there are several ways to change the FPGA configuration: from full configurations where the entire FPGA configuration memory is modified, to partial runtime dynamic reconfiguration were only a piece of the FPGA array is modified while the remaining of the devices is running. The last reconfiguration method is very attractive for the current work as it will permit to modify on the fly the custom HW block. However, partial reconfiguration in Virtex II Pro FPGA has a main restriction. The reconfiguration atomic unit is a frame that spans the entire FPGA high, from the FPGA top IOBs to the bottom IOBs. This restricts the reconfiguration zones that can be defined to complete FPGA columns. This restriction is very important and it will be refereed again afterwards.

The availability of an embedded processor along with the reconfigurable array makes this FPGA a suitable to be used as a first approach for the implementation of the presented in the previous section idea. However, some restrictions have been found during the implementation process that will be further discussed. Next, the focused will be put on the implementation of the system reconfigurability.

In order to implement the reconfiguration related aspects of the presented approach on this commercial device, it is necessary: first, to design a partial reconfigurable system with one reconfigurable module that will allocate the custom HW fabric, second to provide some system in charge of controlling the reconfiguration process along with the configuration library.

In order to design a partial reconfigurable system, first the fixed region of the FPGA has to be clearly defined. In this case, this region will hold the entire PPC system, with all the peripherals (external memory and serial interface-UART), as well as the system for reconfiguration control and the profiler. A further aspect is the allocation in the FPGA array layout of this fixed region, and as it involves interconnections with FPGA external devices, it depends on the selected target board. The selected board is Xilinx University Program Virtex II Pro (XUP Board) development board that is widely used due to its reduced price. The XUP board has a wide variety of peripherals connected to the Virtex II Pro FPGA: a DDR memory slot were up to 1 GB of memory can be plugged slot (currently a 512 MB memory is used), a system Compact Flash (CF) memory interface that is usually used to hold FPGA configurations, Video (XSGA) and Audio (AC97) output interfaces, keyboard, mouse, Ethernet and serial UART interface. From all this interfaces the ones that are wanted to be used are: the DDR memory, the compact flash and the serial UART for software debug.

After an analysis of the allocation of this peripheral to FPGA IOBs banks, it is clear that defining a specific fixed and localized region is not an easy task. The external memories, DDR and CF are allocated in the FPGA layout left side. The

Ethernet and serial UART are allocated in the top part, while the FPGA ICAP port is physically allocated in the FPGA right bottom corner. Furthermore, this FPGA has two PPCs embedded in the reconfigurable array as it is shown on Figure 3. Consequently, there is no clear "free" space in the device that can be used to allocate the reconfigurable module. Therefore it has been decided to restrict the use of the board CF card and allocate the reconfigurable module in the bottom central part of the FPGA (see Figure 3.). The remaining region of the FPGA is defined as the fixed area. The complete, layout of the FPGA that has resulted can be seen on Figure 3, where all the peripherals that can be used are shown.

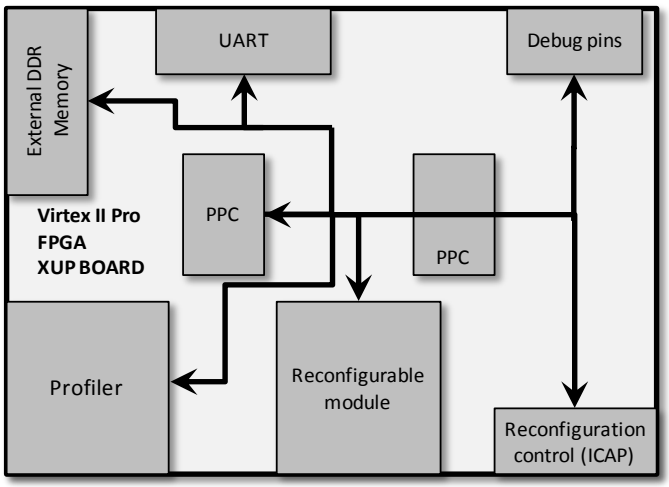

Figure 3. System mapped on the Virtex II Pro FPGA of an XUP Board

As it has been mentioned, the on board CF memory has been restricted and it can not be used. Therefore the configuration library will be held into the board DDR memory. The remaining elements of the reconfiguration control system are: the ICAP controller and a SW tool in charge of the proper allocation of FPGA configurations.

The ICAP controller is a standard Xilinx peripheral component that has been added to the system. In order to control this peripheral, Xilinx provides a driver, called XPART ICAP Driver. The driver permits to load a partial configuration file into the FPGA, to readback a piece of the configuration and kept the read data into a memory. Nevertheless, the provided driver does not solve the problem of the proper allocation of the configuration data. Partial configuration files contain already placed and routed designs that have a specific position in the FPGA and as it has been mentioned, in Virtex II Pro FPGAs reconfigurable modules have to span the entire FPGA. Therefore, in order to obtain a layout like the previously described, with a square reconfigurable module, a tool called pBITPOS has to be used. This tool, intended for embedded systems and valid for Virtex II Pro FPGA, has been previously presented in [8] and its main functionality is to merge two partial configuration files into a single one in order to be able to reconfigure modules that do not span the entire FPGA high, like in this case.

Configuration files to be loaded in the reconfigurable module are held in the local configuration library that is allocated into the XUP Board DDR memory. These files have been generated using the Xilinx conventional design flow based on the Integrated System Environment (ISE) tool that generated complete configuration files. Afterwards, a tool called BITPOS [9] has been used to extract the configuration data related only to the reconfigurable modules and save them into the configuration library.

3.2 Implementing Profiling Block on FPGA

The focus in this work is an implementation of a small profiling block in hardware designed to detect *for* loops in a Virtex II Pro system. A study of compiler code for PowerPC405 core demonstrates that for a loop it generates a non conditional branch backward instruction with some address offset. By decoding the instruction the address of the loop's beginning and size of the loop body is extracted. Additionally we use a counter for the number of iterations in every analyzed loop so each time the loop is executed the value of the counter increases. It is also very important to know not just the number of times loop is executed but the execution time per iteration so a timer is used to count the cycles between interactions. Combining both parameters a more complete data for the best candidate to implement in hardware is evaluated. The first and most important measure is the loop frequency which is stored in counter for every loop we

analyze. When a non conditional brunch is detected first a check if it is already observed is done and if so the counter is increased. A timer starts and when the next iteration of the same loop occurs it is compared to the old timer value and if it's the same value or less it is stored again. If timer value is bigger than the previous it is skipped in order to keep always the minimum time as a more restrictive measure for evaluation of the best block to be implemented in hardware. In our implementation we analyze ten loops replacing less interesting candidates by applying a weight factor. In order to choose best candidates we use iteration time and frequency of the loop. By combining these two parameters and keeping the weight factor as a threshold value for the next candidate we guarantee that the most frequent and time consuming loops will be selected. In the best case scenario our block will be connected to the PowerPC data cache memory directly but due to architectural restrictions this is not possible so we are connected to a block RAM witch stores the application under profile. In a future implementation a soft processor can be used to gain direct access to the cache modifying the processor cache controller. After execution profiling data is recovered by UART connection to a host PC so a further analysis can be done.

The instruction that is decoded by the hardware block was designed as a generic parameter witch alloys to change the type of instruction under monitoring so instead of for loops the designer can trace function calls. This feature actually makes very simple the change of granularity of the analysis and allows comparing different approaches for accelerating.

4. TEST AN RESULTS

The fine grain code detector block used for implementation tests and validation tracks data from ten loops. This block has been synthesised with the Integrated System Environment (ISE 9.2) tool with time optimizations.The resulting FPGA area requirements of the block can be found on Table 1. The number of loops to be under analysis could be increased causing increased block area.

Table 1. Hardware Profiler Logic Utilization of the target FPGA XC2VP30

Logic Utilization	**Used**	**Available**	**Device Utilization**
Number of Slices	443	4928	9%
Number of Slice Flip Flops	315	9856	3%
Number of 4 input LUTs	841	9856	9%

The functionality of the block has been verified by using a standard benchmark, called MediaBench [10]. This is well none public domain benchmark suite focused on multimedia and communications for embedded systems. It includes a complete set of multimedia and communications standards such as MPEG, JPEG, GSM and G.721.one of the most calculus demanding kernel in the suite and the selected one is an MPEG video component called *mpeg2dec*. This benchmark component is executed on FPGA´s PPC including a standalone file system (XilLib) in order to access input and output video streams. After a performance analysis, it has been noticed that MediaBench component *mpeg2dec* more than 80 percent of the execution time is spend in small loops. In that particular benchmark kernel 114 *for* loops in total were detected and with our profiling block gathering information about ten loops we cover 8.8 percent of the loops in that particular application. After running the benchmark candidate blocks are stored on the host and compared manually to high level code in order to verify the block functionality.

One of the limitations of the profiling block is that loop detection is done on high level code with no compiler optimization involved.

5. CONCLUSIONS

Though limited by several factors this implementation of hardware loop detection demonstrates that with relatively small area coast loop detection as a first approach to hardware acceleration is possible. One of the main limitations is the access to the processor´s cache memory. If a different architecture is used such as Leon [12] soft processor the profiling block could be embedded in the processor architecture and effectively gather information about small hardware blocks within time consuming blocks in the software.

REFERENCES

[1] Susan, L., Graham P., B., Kessler and M., K., McKusick, “Gprof: A call graph execution profiler ,” ACM SIGPLAN Notice 17(6), : 120 – 126 (1982).

[2] Vassiliadis, S., Wong, S., Gaydadjiev, G., N., Bertels, K., Kuzmanov, G., and Panainte, E., M., "The molen polymorphic processor, " IEEE Transactions on Computers, pp. 1363- 1375, November 2004.

[3] Gordon-Ross, A. and Vahid, F., "Frequent loop detection using efficient nonintrusive on-chip hardware, " IEEE Transactions on Computers, 54(10),1203-1215(2005).

[4] Ung, D.and Cifuentes, C., "Dynamic re-engineering of binary code with run-time feedbacks, " Proceedings on Seventh Working Conference of Reverse Engineering,2-10 (2000).

[5] Lysecky, R., Vahid, F., Tan, and S.D.-X., "A study of the scalability of on-chip routing for just-in-time FPGA compilation," 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 57-62(2005).

[6] Matev, V., de la Torre, E. and Riesgo, T., "Towards Fine and Medium Grain Dynamic Functional Extraction for HW/SW Acceleration," 3rd Southern Conference on Programmable Logic, SPL. , 93-98 (2007).

[7] Xilinx Inc, “Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Functional Description DS083 (v4.7)”, data sheet, (2007).

[8] Krasteva, Y., Joly, D., de la Torre, E., and Riesgo, T., ”Virtex II Bitstream Manipulation: Application to Reconfiguration Control Sytems ”, Proceedings of 16th IEEE Intl. Conference on Field Programmable Logic and Applications, 1-4 (2006).

[9] Krasteva, Y., Jimeno, A., B., de la Torre, E., and Riesgo, T.,”Straight Method for Reallocation of Complex Cores by Dynamic Reconfiguration in Virtex II FPGAs, ” Proceedings of the IEEE Intl. WorkShop on Rapid System Prototyping (RSP), 77 – 83 (2005).

[10] Chunho Lee, Potkonjak, M., Mangione-Smith, W.H., "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems, " Proceedings Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture, 330-335 (1997).

[11] Xilinx, “Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet, ” Technical report (2005).

[12] LEON:Free Hardware and Software Resources for System on Chip
http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53