International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 4, April - 2014

Survey on Methods To Reconfigure The Instruction Set Processors

Parvathy Asokan
M.Tech
CMRIT, Bangalore

Kavitha.V
Ph.D Scholar,
Jain University

***Abstract* – In this paper, we are comparing two techniques to reconfigure the Instruction Set Processors: Dynamically Reconfigurable RISP and Expression Grained reconfigurable Array. In Dynamically Reconfigurable RISP, the reconfiguration can be done during runtime. For processors with Expression Grained reconfigurable Array (EGRA), the processor is stalled during reconfiguration.**

Keywords- Dynamically Reconfigurable RISP, Expression Grained Reconfigurable Array(EGRA)

I. INTRODUCTION

General Purpose processors (GPP) are designed for general purpose computers. The most important concern of GPP is the computation speed. GPPs can be used in embedded systems because of their features such as flexibility, low cost design, programmability, tools availability, less time to market etc. The drawbacks of GPPs are their inefficiency for high performance computing. In order to accelerate the performance of GPP, an application specific instruction set extension can be added to the basic processor. Such a processor is called Application Specific Instruction Set Processors (ASIPs). In ASIP, the critical portions of the application can be executed on Custom Function Unit (CFU).Reconfigurable Instruction Set processors (RISP) consists of a microprocessor core that has been extended with the reconfigurable logic. It is similar to ASIP but instead of CFU, it contains RFU. The design of a reconfigurable processor can be divided in two main tasks. The first one is the interfacing between the microprocessor and the reconfigurable logic. The second task is the design of the reconfigurable logic itself. In RISP, the Instruction Set Architecture (ISA) can be expanded during runtime after manufacturing. They provide trade off between efficiency and flexibility. There are different techniques for reconfiguring instruction set processor- Reconfiguration during runtime, Reconfiguration by stalling processors etc. in this paper we are comparing two techniques: Dynamically Reconfigurable RISP and Expression grained Reconfigurable Array (EGRA).

II. RELATED WORK

There are several techniques to reconfigure the instruction set processors. Authors in [4] demonstrate that combination of multicore processor and reconfigurable instruction set extensions creates multi- level parallelism for high performance. ReMAP (Reconfigurable Multicore Acceleration and Parallelization) [5] uses common RFU between heterogeneous cores. RFU is loosely coupled with cores with a fine granularity. The methodology in [6] customizes a MPSoC platform by a repetitive procedure. Initially it assigns the tasks to processors and then adds CIs for the tasks which are on critical path until the selected path is no longer critical. EGRA is inspired by the Configurable Computation Acceleration (CCA) structure proposed by Clark [7]. The CCA is used as standalone acceleration. The authors in [8] describe a pipelining scheme for EGRA.

III. DYNAMICALLY RECONFIGURABLE RISP

In Dynamically Reconfigurable RISP, reconfiguration can be done during runtime. The instruction set of RISP is not fixed during design time. It is essential to change the manner in which code for RISP is generated. RISP code generation involves code generation techniques and hardware design techniques. The basic element of the code generation for a RISP is the High Level language (HLL) compiler. The HLL compiler will do the hardware/software partitioning. The fig.1.shows the generic RISP compiler flow

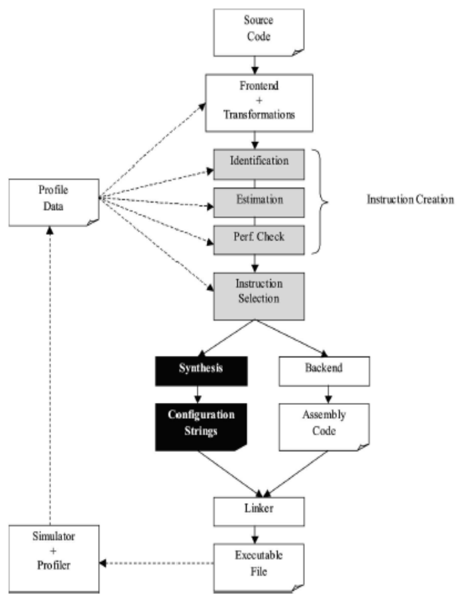

Fig.1. Generic RISP Compiler Flow [1]

In the figure, the white blocks represent traditional compiler blocks and tools. The grey area represents new techniques for RISP and black areas represent traditional hardware synthesis technique. The source code is analyzed and an internal representation in the form of control and data flow graph is obtained. This graph consists of basic blocks, hyper blocks or constructs that facilitates compilation. The blocks size depends on the architecture of the processor. During these stages, high level optimization steps are also done.

After obtaining the control and data flow graph, the next step is the *identification of instructions* to be implemented in RFU. This depends on the internal representation used. There are two techniques for instruction identification.

- Data flow/ general techniques
- Ad hoc/ customized techniques

In the data flow technique, groups of operations can be combined to obtain a complex instruction. The Ad hoc/ customized techniques are specialized techniques. In this technique, special construct in the application is identified and create a new instruction specifically for that.

An example for a complex instruction is shown in fig.2.

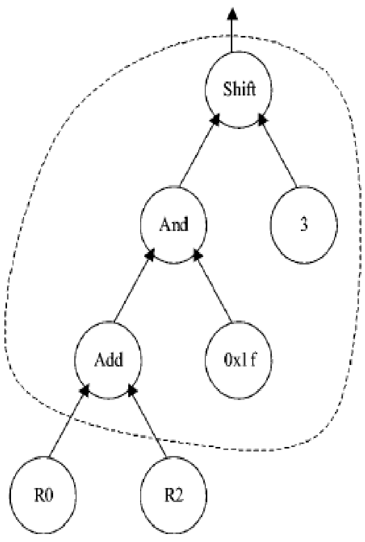

Fig.2. Generation of a complex instruction [1]

The next step in the generic RISP compiler flow is the instruction characterization. This is done to obtain the estimate of the required parameters without doing complex synthesis. This can reduce the compilation time. By doing this, we can obtain the instructions which do not fit inside the RFU. Such instructions are discarded.

After instruction creation and characterization, the instructions can be checked to determine whether there is any increase in performance. If not, the instruction is rejected.

The next step is *Instruction Synthesis*. It takes the description of an instruction and generates the configuration string for the RFU.

The backend of the compiler performs platform specific optimizations and outputs the assembly code. There are three main optimization techniques.

- Manual identification: The programmer elucidates the code with special compiler directives and identifies the places where the compiler should optimize.
- Static identification: the compiler analyzes the code and identifies the code for optimization.
- Dynamic identification: The code is compiled initially without optimization and then the code is profiled to obtain the places of optimization. This is time consuming but can achieve better results.

The main advantage of this technique is that this can lead to solutions in which the processor spends most of its time reconfiguring the RFU. This also reduces the power consumption.

The main issue faced by the Dynamically Reconfigurable RISP is related to code generation. It is also difficult to manage the reconfiguration delay.

IV. EXPRESSION GRAINED RECONFIGURABLE ARRAY (EGRA)

Expression Grained Reconfigurable Array consists of an array of cells consisting of a group of Arithmetic logic Units (ALUs) with customizable capabilities. These array of cells are called Coarse Grained cell RAC (Reconfigurable ALU cluster). The architecture which embeds it is called Expression Grained Reconfigurable Array. The RAC is the heart of EGRA. It supports efficient computation of entire sub expression. In EGRA, each cell consists of a cluster of ALUs while the CGRA consists of a single ALU. So the CGRA can perform single operation only. The EGRA removes the limit on the number of inputs and outputs. The comparison between EGRA and CGRA is shown in fig.3

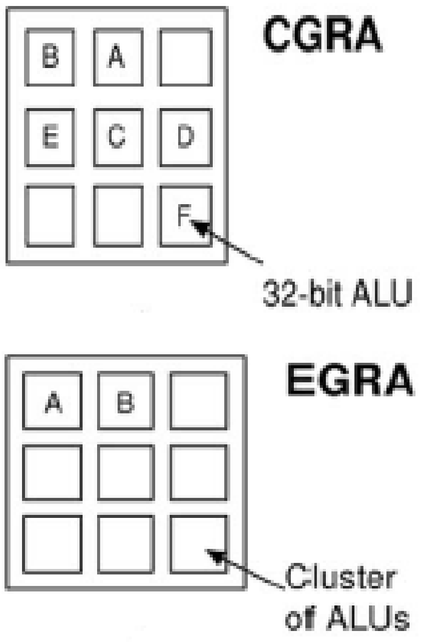

Fig.3. Comparison between EGRA and CGRA

The data path of RAC is shown in fig.4.

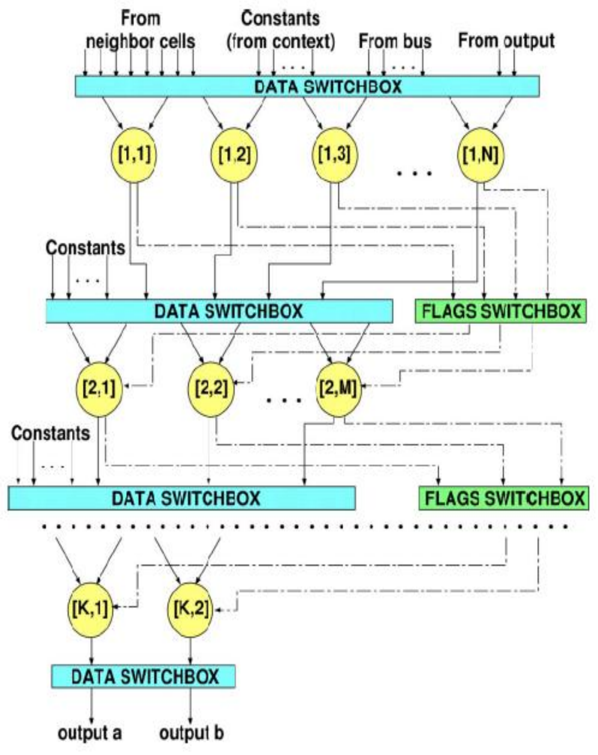

Fig. 4. Data path of RAC [2]

The ALU in each cell are organized as rows connected by switch boxes. The inputs of the RAC are obtained from the neighboring cell output or from the output of the cell itself or from a set of constants

The RAC of EGRA consisting of multiple ALU cluster, memories and multipliers are shown in fig.5.

There are two operational modes for EGRA.

- DMA mode: DMA mode is used to transfer data in burst to the EGRA and to program the cells and to read/write from scratchpad memories.
- Execution mode: In execution mode, the control unit controls the data flow between the cells.

It is possible to interface the EGRA with the extensible host processor. Such type of processors supports variable latency custom instructions. When EGRA is executing, the host processor is stalled until the EGRA completes its operation, and asserts a “done” signal.

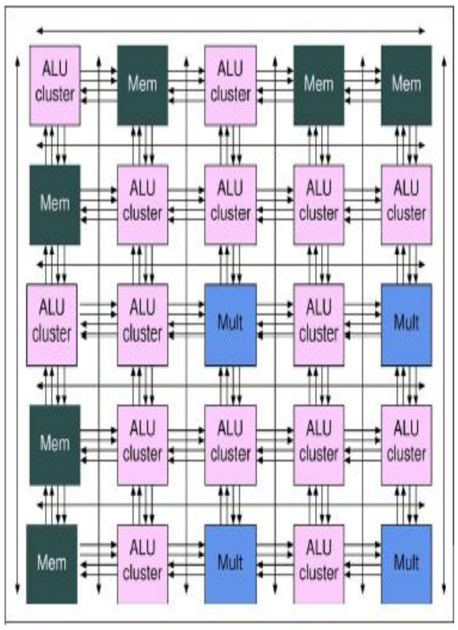

Fig.5. EGRA instance example: a 5 x5 mesh with 15 RACs, 6 memory cells, and 4 multipliers [2]

The main advantage of using the processor with EGRA is the stalling of processor while EGRA is executing. So the power consumption is reduced. The processor with EGRA also reduces the completion time of an application and hence it increases the performance. But the drawback is the increase in area.

V. CONCLUSION

In this paper, we compared two techniques for reconfiguring instruction set processors: Dynamically Reconfigurable RISP and Expression Grained Reconfigurable Array (EGRA). From the investigation, we obtained that EGRA is having less power consumption and high performance as compared to Dynamically Reconfigurable RISP. This is because for the processor with EGRA, the processor is stalled while the reconfigurable logic is executing.

VI. ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to my Guide **Mrs. Kavitha.V**, Associate Prof, Electronics And Communication Engineering, CMR Institute of Technology, for her timely advice on the technical support and regular assistance throughout the project work.

REFERENCES

[1] F. Barat, R. Lauwereins and G. Deconinck, “Reconfigurable instruction set processors from a hardware/ software perspective”, Software engineering, IEEE Transactions on, vol.28, no.9, pp.847-862, 2002.

[2] Giovanni Ansaloni, Paolo Bonzini, Laura Pozzi, “EGRA: A Coarse Grained Reconfigurable Architectural Template”, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on Vol.19, no. 6, pp 1062 – 1074, 2010.

[3] F. Barat and R.Lauwereins, **“**Reconfigurable Instruction Set Processors: A Survey”, Rapid system prototyping, 11th international workshop, 2000, pp.168- 173.

[4] Z. Chen, R. N. Pittman, and A. Forin, "Combining multicore and reconfigurable instruction set extensions," in *Proceedings of the 18th annual ACMISIGDA international symposium on Field programmable gate arrays,* 2010, pp. 33-36.

[5] M. A. Watkins and D. H. Albonesi, "ReMAP: A reconfigurable heterogeneous multicore architecture," in *Micro architecture (MICRO),2010 43rd Annual IEEEIACM International Symposium on,* 2010, pp.497-508.

[6] F. Sun, S. Ravi, A. Raghunathan, and N. Jha, "A Framework for Extensible Processor Based MPSoC Design," *Designing Embedded Processors,* pp. 65-95, 2007.

[7] N. Clark, M. Kudlur, H. Park, S. Mahlke, and K. Flautner, “Application-specific processing on a general-purpose core via transparent instruction set customization,” in *Proc. 37th Ann. Int. Symp. Micro arch. (MICRO’37)*, Washington, DC, Dec. 2004, pp. 30–40.

[8] L. Pozzi and P. Ienne, “Exploiting pipelining to relax register-file port constraints of instruction-set extensions,” in *Proc. Int. Conf. Compilers, Arch., Synth. Embed. Syst.*, San Francisco, CA, Sep. 2005, pp. 2–10.