Soft-core reduction methodology for SIMD architecture: OPENRISC case study

Bouthaina DAMMAK
CESlab
Sfax University (ENIS school)
Sfax,Tunisia
bouthaina.damak@gmail.com

Mouna BAKLOUTI and Mohamed ABID
CESlab
Sfax University (ENIS school)
Sfax,Tunisia
baklouti_mouna@yahoo.fr

***Abstract*—Multi-Processor Systems on Chip (MPSoCs) have been proposed as a promising solution for the increasing demand of computational power required for recent application. The parallelization through SIMD (single instruction/multiple data) architectures has been a proven solution to speed up the processing of the recent application that exhibit massive amounts of data parallelism. The level of parallelism impacts the SIMD architecture performance and it is closely related to the design of the processing element. In this context this paper presents a new design methodology of designing processing element for SIMD architecture. The scope of this work is to reduce the pipeline stages of the soft-core processor to reduce the size of the PEs and so that to built up a high level parallelism architecture.**

I. Introduction

SIMD [1] represents one of the earliest styles of parallel processing. The term SIMD stands for "Single-Instruction Multiple-Data," which aptly encapsulates the parallel processing model. These architecture have established themselves as a suitable solution for a wide range of a highly data parallel applications [2]; they are geared toward applications that exhibit massive amounts of data parallelism without complicated control flow or excessive amounts of inter-processor communication. Typical applications for SIMD machines include low-level vision and image processing, discrete particle simulation, database searches, and genetic sequence matching.

Closely related to vector processing, the basic idea is to operate the same instruction sequence simultaneously on a large number of discrete data sets [1]. To achieve this functionality, a SIMD machine basically consists of an array of fine-grained computational units connected together in some sort of simple network topology [3]. This processor array is connected to a control processor (ACU), which is responsible for fetching and interpreting instructions. The ACU issues arithmetic and data processing instructions to the processor array, and handles any control flow or serial computation that cannot be parallelized [1].

To achieve their performance the individual processing elements are usually very simple in nature and targeted to the application for which they are designed [4]. Within this methodology the PE is built around the necessary unit of the application: Multiplication unit, comparator unit, registers, etc [4]. The small size of PEs within this methodology allows a high integration capability and so that a high performance. The drawback of this method of conception is the difficulty of conception and, long time required to designing and the restriction of the built PE to the given application. Therefore, with the advance made in designing soft-core [5] processor a new design methodology based on the use of available IP processor is appeared. In this contest, this project aims to propose a processor design reduction methodology based on standard processor IPs in order to build high parallel processor architecture. The idea is to minimize the logical area utilization of the PE in order to integrate a maximum number into the architecture. To do so the PEs pipeline stages are reduce. This reduction concerns the pipeline stage of the Soft-core processor. The fetch and decode stages of the PEs are eliminated and the Array Control Unit (ACU) is the responsible of handling the microinstructions to the PE array.

Today the FPGA presents an ideal platform for a massively multiprocessor architecture prototyping due to their inherent integration capacity. Therefore to test and evaluate the proposed methodology a case study will be proposed and an implementation test based on Altera Stratix II FPGA will be performed.

The remainder of the paper is organized as follows. Related work of designing processing elements is presented in the second section. We describe our proposed methodology in the third section. The fourth section presents a study of the soft-core processor. Implementation results and analyses of the developed methodology are shown in the fifth section. Finally, conclusions are given in the sixth section.

II. Related Work

The impact of machine structure on system performance is a critical consideration in designing highly integrated SIMD architectures. This issue is highly affected by PE granularity and PE complexity. To meet this need, SIMD machines are usually based on the use of simple PEs designed to the application for which the architecture is built.

In reference [6] Takashi Komuro & al have proposed a PE architecture called S3PE (Simple and Smart Sensory Processing Elements) targeted to vision algorithm. The PE architecture consists of an ALU, local memory and three registers. This simple PE architecture has a general purpose utility and permits a high integration level on a chip. But the function of the ALU and the size of the local PE memory are not enough for recent complicated visual application.

Reference [7] proposes VBMSE architecture to compute the motion estimation vector of a video frame for the H.264/AVC. This architecture is composed of a systolic array of regular data processing elements (PE). Each PE is composed of three units: Fixed Block (FB), Comparator Unit (CU) and the Reuse Unit (RU). The architectures of these units are well organized to produce in a final step the value of the *SADMIN.*

Reference [8] proposes the morphosys architecture targeted for image processing applications. It is composed of a reconfigurable array, a control processor, a data buffer and a DMA controller. The reconfigurable array is an 8 by 8 array of reconfigurable cells operating in SIMD fashion and constitutes the processing element array. Each cell has an ALU-multiplier, a shift unit, and two multiplexers for ALU. The RC Array functionality and interconnection network are configured through 32-bit context words.

The [7] and [8] reference architectures are more efficient than [6] architecture but their PE design is targeted to the desired application.

This method of conception usually responses to the required performance but faces the problem of difficulty and a long time of conception as well as a limitation of utilization for a specific application. Therefore, with the efforts made in designing Soft-core processor which provide sufficient performance with less cost, a new design methodology based on the use of Soft-core processor was appeared.

Reference [9] proposes a parallel processor architecture that is based on the use of the picoblaze processor. This architecture contains the following core elements: a processor field composed of 1-bit processing elements (PEs), the control unit and an image buffer.

The PE of this architecture is a reduced picooblaze from Xilinx. Some of the characteristics of the PicoBlaze are the 16 registers, the 64-byte internal RAM, the CALL/RETURN stack and the on-chip program memory with 1024 instructions implemented as BRAM. The PE is built as a reduced Picoblaze. The reduction included the arithmetic operations, the shift and rotates instructions, the interrupt handling as well as the internal memory. All this leads to a reduced PicoBlaze which needs a few number of slices. The limitation of this method is the reduction of the processor instructions which limits the application complexity.

III. Reduction Methodology

As mentioned in the previous section the use of personalized methodology in designing PEs limits their use to the applications for which they are designed and requires a relatively long time for conception.

Therefore IP based methodology has been proposed to tackle the problem revealed from the use of personalized PEs. And in order to overcome the relatively high logic utilization raising from the use of soft-core processor, our idea is to reduce the pipeline stages which consist in eliminating the fetch and decode stages. In fact, the ACU is responsible for fetching and decoding the instructions and issues parallel microinstruction treatment to the PEs array. To ensure the scent of the parallel microinstruction to the PEs array, the ACU will be built up as a modified Soft-core processor. Our proposed methodology will be splitted into five steps

A. First step : processor test

After downloading the processor it is necessary to study it (component, simulation environment, etc).

This processor could be either distributed standalone or available in a soc distribution. The first distribution only involves the soft-core processor which should be connected to an on-chip RAM in order to test and simulate it.

The second distribution is a collection of extra-modules like RAM, UART, Ethernet and VGA. Some examples of this distribution are the leon2 and the OpenRISC processor.

To test and simulate the processor, for the standalone distribution, the Soft-core must be connected to a RAM. For the soc distribution the optional modules can be eliminated from the soc top-level.

B. Second step: Processor optimization

The processor optimization involves two steps. The first step consists in the optimized processor configuration; the processor configuration is made through a configuration file which contains the definition of several parameters. The implementation of each of the module depends on its parameter definition.

The second step is to check the possibility to eliminate complementary modules which are not included in the configuration file such as the Tick timer. Once optional modules are eliminated and in order to ensure the processor functionality, it is necessary to check the processor interface; eliminate the I/O interface signals related to the eliminated modules.

C. Third step : Study of processor components

To success the ACU and PE design, the functionality of the processor modules must be understood and then associated to its pipeline stage. Then a specific focus must be given to the connection between the fetch/decode stage and the execution stages: the direction of all input/output signals of the fetch and decode stage modules must be studied.

D. Forth step : ACU and PE building

This step requires specific concern as well as high understanding level of the third step.

1) ACU building

As the ACU is responsible for the handling of micro-instruction to the PEs, it is necessary to modify the processor interface by adding an I/O interface that is intended to communicate with the PE. To develop this concept of ACU building, we must proceed as follow:

a) Allocate specific instructions at the start of each type treatment. So that the ACU will be able to differentiate between parallel and sequential treatment.

b) Adding a test process sensitive to the RAM output data within the decode module, that allows the decode stage to recognize the type of treatment; this test compare each new instruction to the specific instruction of the sequential and parallel treatment.

c) Rectify the fetch and program counter module; the fetch and pc counter module signals that are connected to the execute stages (gray dotted fetch and gen-pc signals in Fig.1) must be declared as global I/O (gray fetch and gen-pc signals in Fig. 1) of the ACU to ensure the communication of the fetch stage with the PE execute stages.

d) In the decode block, a new I/O interface is added (Fig.1). The signal definition of this interface is similar to the initial communication signals between the ACU decode and execute stages with the difference that these signal are activated for a parallel treatment whereas the initial signals are activated in the case of sequential treatment. Therefore the implementations of the initial signals are modified: they are activated if the result of the test indicates a parallel treatment otherwise they are disabled.

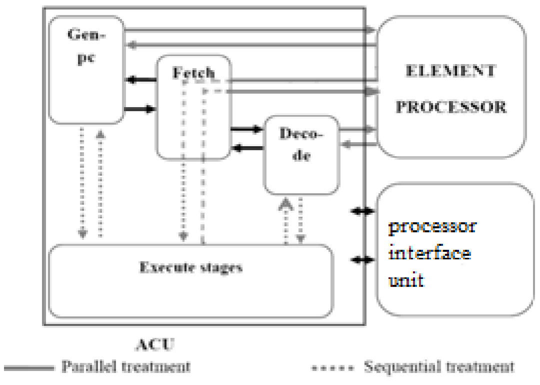

FIGURE 1. *ACU DESIGN*

2) PE building

To ensure the ACU-PE communication an I/O interface must be created. The input created interface actually contains the ACU decode and fetch pipeline module outputs that are intended to be connected to the execute stages of the PE (studied in the third step). The communication signals between the soft-core output fetch/decode modules and the execute modules ought to be replaced by the new created interface signals (Fig. 2). The output created interface contains the ACU fetch and decode stages inputs and intended to be connected the PES (studied in step 3). This output interface will be driven as input to the ACU in order to ensure the running of its decode and fetch stages in case of parallel treatment (Fig. 2).
Getting all these modifications established the fetch and decode block must be eliminated from the processor top level and so the PE build up is accomplished.

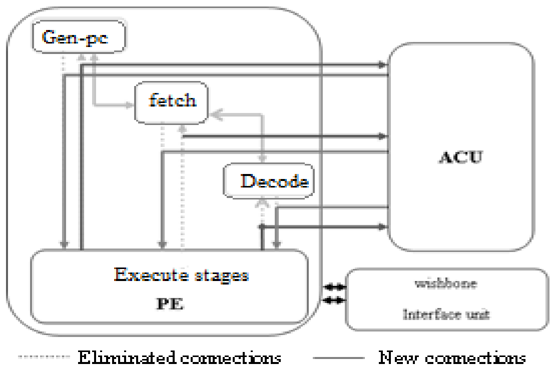

FIGURE 2. *PE DESIGN*

E. Fifth step : Design Test

At this final step it is intended to test the running of the ACU and PE in order to approve the built system approach.
The test procedure consists in:

- Establishing the connection between the ACU and a number of PEs.
- Simulating the system using a parallel code.

The test should result in recording the results of the executed program in the PEs memory. If the review of the recorded data find out unexpected results or the record has not been done, the ACU-PE communication or the PE /ACU building must be verified.

IV. CASE STUDY

A. Overview

To choose the suitable soft-core processor to test the developed methodology, a comparative study of three soft-core processors was achieved: LEON2 from Gaisler Research, MicroBlaze from Xilinx and OpenRISC 1200 from OpenCores was established.
The OpenRISC was chosen for its performance and its less logical area utilization
The OpenRISC 1200 [10] is written in the Verilog hardware description language. It is a 32-bit scalar

RISC processor with Harvard micro architecture, wishbone interface, five bit integer pipeline, virtual memory support (MMU) and basic DSP capabilities [11]. The tool chain of the OpenRISC requires the or1K_binutils, or1K_gcc, or1K_gdb and the or1K_sim. The C can be cross compiled and the hex file is obtained.

B. *OpenRISC CASE STUDY*

The Case study was based on a SoC implementation based on the OpenRISC 1200 processor, known as ORPSoC (the OpenRISC Reference Platform System-on-Chip).The steps of the developed methodology (section III) was applied to the ORPSOC design. The ORPSOC design was optimized and the Soft-core pipeline stages were reduced. The specific instructions used to differentiate parallel and sequential treatment are l.add r7,r7,r0 (for the parallel treatment) and l.addi r7,r7,0(for the sequential treatment). A SIMD architecture based on the reduction methodology was built and was tested with different assembler and c code. The next section will present the experimental results of the reduction methodology applied to the OpenRISC processor.

V. EXPERIMENTAL RESULTS

Within this section, it is intended to evaluate the implementation results and the execution time efficiency of the SIMD proposed architecture.

The working strategy begins with the ORPSOC design optimization and an FPGA StratixII based implementation .Then the implementation results of the SIMD proposed architecture and the replication methodology is performed. And finally the evaluation of the execution time of the proposed architecture resulted of different parallel algorithms is performed.

A. *Design optimization*

In order to minimize the logic utilization of the Soft-core processor design, the optimization task is essential and an implementation results comparison of the initial design and the optimized one will be informative. The extra-modules were eliminated and the optimal processor configuration was achieved. The functionality of the optimized design was tested and verified. The particular FPGA used for the test was an Altera StratixII-2S180. This chip is made up of an array of 180 000.

TABLE 1. IMPLEMENTATION RESULTS OF THE OPTIMIZED ORPSOC DESIGN

	The initial design	Optimized design
Device	EP2S180F1020 C3	EP2S180F1020 C3
Logical use	5%	3%
Combinational ALUTs	6396/ 143 520	3524/ 143 520
Openrisc logic utilization	5904	3338

The table 1 shows that the design optimization greatly affects the logic utilization which decreases from 6396 Luts for the initial design to 3524 Luts for the optimized one. The suitable OpenRISC optimization results on a decrease of the OpenRISc processor utilization from 5904 to 3338 Luts.

In order to proof the utility of the PE pipeline stage reduction, a comparison between the reduction methodology and a replication method will be presented. The replication is based on a collection of the soft-core processor. The only modification is made to handle the instructions from the ACU to the PEs; The PEs decode stage receives the instructions from the ACU.

B. *Comparison between the two methodology implementation results*

To obtain architecture with a high parallelism level, the PEs number was increased for each one until the design could no longer fit on the FPGA. The table 2 shows the implementation results of the replication and reduction methodologies on FPGA.

TABLE 2. IMPLEMENTATION RESULTS OF THE REPLICATION AND REDUCTION METHODOLOGY

	Replication methodology	Reduction methodology
2 PEs	8%	6%
8 PEs	28%	17%
24 PEs	78%	42%
32 PEs	Can't fit on FPGA	58%
40 PEs	Can't fit on FPGA	72%

The first observation raised from table 2 is that the maximum number of implemented PEs to each architecture is different. For the replication methodology the maximum number was 24 with 78 % and 40 PEs within the reduction methodology with 72%.

The second observation is that the augmentation average of the logical area utilization between the two methodologies is different (Fig. 3). The slope of the red curve is much more important than the blue one.

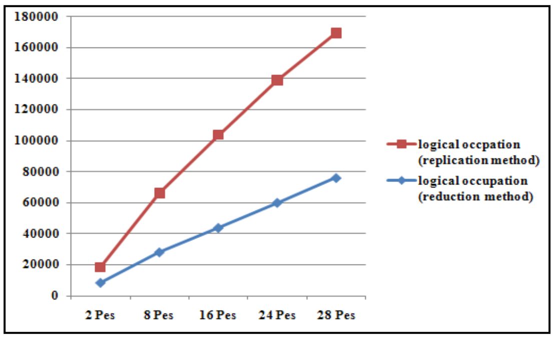

FIGURE 3. *EVOLUTION OF THE LOGICAL OCCUPATION*

This difference is the result of the difference of the PE logical size within each methodology. The table 3 shows the implementation detail of a SIMD architecture composed of an ACU and two PE within the replication and reduction methodology.

TABLE 3. IMPLEMENTATION DETAILS WITHIN THE REDUCTION AND REPLICATION METHODOLOGY

	Replication methodology	Reduction methodology
ACU logic utilization	3564	3117
PE1 logic utilization	3512	2789
PE2 logic utilization	3514	2751

The table 3 shows the difference of the PEs size within the two methodolThogies. The reduction of the pipeline stages and so that the elimination of the fetch and decode modules effects greatly the size of the PE within the reduction methodology.

C. *Execution time performance*

The previous section revealed the advantages of the developed methodology in term of logic occupation. The present section aims to evaluate the execution time performance of the developed methodology. To do so, two algorithms are selected; Matrix-vector multiplication and the correlation application.

The execution of these parallel algorithms will be performed with a 100 MHz frequency.

1) Matrix-vector multiplication

a) Overview

We consider a square matrix of size n and X a vector of size n and Y is the multiplication resulting vector. The element i of the resulted vector is the multiplication of the line i of the matrix A with the vector X. The parallelization of the multiplication consists in dividing the calculation of the Z elements among the PEs. If the parallel architecture is composed of p PEs, each PE uses N/p lines of X to calculate N/P elements of Y (Fig. 4).

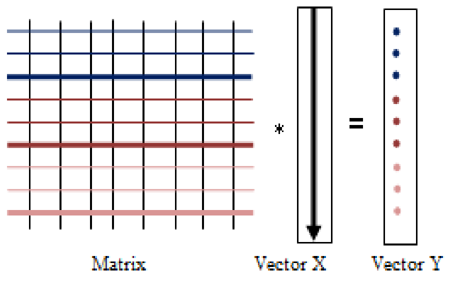

FIGURE 4. *PARALLEL MATRIX-VECTOR MULTIPLICATION*

b) Results

The features of the matrix and vector used in the Matrix-vector multiplication running are:

Matrix A(128,128) : $\begin{pmatrix} A00 & \cdots & A0\ 128 \\ \vdots & \ddots & \vdots \\ A128\ 0 & \cdots & A128\ 128 \end{pmatrix}$

Vector Y: $\begin{pmatrix} Y1 \\ . \\ . \\ . \\ Y128 \end{pmatrix}$

The serial algorithm was executed on the OpenRISC processor with 19 619 us. Then the parallel c code was running on SIMD architecture with different PEs number. The serial and parallel execution time allows the calculation of the speed up which measure the performance gains of the parallel algorithms over the serial algorithm. The table 4 presents the execution time and the Speed up of MPPSOC architecture with different PE number for the Matrix-vector multiplication code.
The table 4 shows the evolution of the execution time with the PEs number increase. This time decrease from 19 619us for serial computation to 5100 us for parallel computation with 32 PEs .The efficiency of the SIMD architecture increase from 1, 5 with 2 PES to 3, 8 with 32 PEs. This speed up is relatively comparable to the results presented in [12] although it is designed from personalized PEs and was programmed with the assembler language .

TABLE 4. MATRIX-VECTOR MULTIPLICATION RESULTS ON SIMD ARCHITECTURE WITHIN THE REDUCTION METHODOLOGY

	Execution time(us)	Speedup
Serial algorithm	19619	
Parallel algorithm	12960	1,5
Parallel algorithm (8 PEs)	7672	2,6
Parallel algorithm (16 PEs)	5644	3,5
Parallel algorithm (32 PEs)	5100	3,8

2) Correlation

a) Overview

The correlation allows for radar to detect an object in its field. The radar sends and receipt signals. The correlation allows, knowing the sent data, to detect objects within the field of the radar. The radar has a code constituted of 1023 coefficient; each one is stored on a one bit. It continuously sends its code and stores the received data stored in 4. The formula to calculate a correlation is:

$$\mathrm{St} = \sum_{k=1}^{1023} C(i) * y(i+t) \quad . \tag{1}$$

If in a given moment we Consider that 64 Y are received from the radar, it's necessary to calculate 64 correlations. Each correlation is calculated as follow:

$$\mathrm{St} = \sum_{k=1}^{1023} C\,(\,i - t \bmod 1023) * y(i). \tag{2}$$

If an object is actually in the field of radar, there is a peak among the 64 correlation results

b) Execution time

The evaluation of the algorithm parallelization requires the calculation of the execution time of the serial algorithm. The code of the serial correlation calculation was written and executed.

The difference between the parallel algorithm and the sequential one is that during iteration the number of correlation calculated is equal to the number of PEs within the parallel architecture while only one correlation is performed during iteration for the serial execution.

Within the parallel treatment, the number of PEs was increased and the execution time and the efficiency related to the architectures was measured (table 5).

The table 5 shows the important decrease of the execution time with the augmentation of the PEs number. This time decreases from 134 409 us for the serial execution to 7 591 us obtained with 32 PEs. The speed up also increases significantly with the PE number augmentation, its value increase from 1, 6 to 29, 5. The architecture is considerably efficient since the speed up is approximately equal to the PEs number.

TABLE 5. CORRELATION ALGORITHM RESULTS ON SIMD ARCHITECTURE WITHIN THE REDUCTION METHODOLOGY

	Execution time(us)	Speedup
Serial algorithm	224 016	
Parallel algorithm (2 PEs)	134 409	1,6
Parallel algorithm (8 PEs)	28 888	7,7
Parallel algorithm (16 PEs)	15 031	14,9
Parallel algorithm (32 PEs)	7 591	29,5

VI. CONCLUSION

The originality of the work described here lies in the use of a new PE design methodology to build up SIMD architectures.

The main difference with other SIMD architectures is the use of a soft-core processor to design the processing elements. In this way the problem of personalized PEs are avoided (restriction of use, difficulty of conception, long time to market). The use of a reduced soft-core ensures a general use of the designed architecture and a high parallelism level. In addition the performance of the soft-core processor permits the conception of an efficient processing element that supports the assembler and c programming language. These benefits have been verified with a case study of the OpenRISC processor.

As expected work, we aim to integrate an interconnection network bus to solve the problem of the ACU and PE memory connection and to run more complicated parallel data application.

REFERENCES

[1] Michael Sung, "SIMD Parallel Processing : Architectures Anonymous", February 22, 2000

[2] http://en.wikipedia.org/wiki/Data_parallelism

[3] Parallel algorithms in Scientific computing Parallel architecture Spring 2009

[4] Parallel and Distributed Processing , Stephen S. Wilson, "NEURAL COMPUTING ON A ONE DIMENSIONAL SIMD ARRAY," 110 Parkland Plaza, Ann Arbor, MI 48103

[5] http://en.wikipedia.org/wiki/Soft_microprocessor

[6] Takashi Komuro, Idaku Ishii, and Masatoshi Ishikawa, "Vision Chip Architecture Using General-Purpose Processing Elements for 1ms Vision System," Department of Mathematical Engineering and Information Physics, University of Tokyo.

[7] H. Parandeh-Afshar, P.Brisk, and P.Ienne ," Scalable and low cost design Approach for variable Block size Motion Estimation, " Ecole Polytechnique Fédérale de Lausanne (EPFL).

[8] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J. Kurdahi, Nader Bagherzadeh, and Tomas Lang, "MorphoSys: An Integrated Re-configurable Architecture," University of California, Irvine.

[9] Frank Schurz, Dietmar Fey and Friedrich-Schiller, "A Programmable Parallel Processor Architecture in FPGAs for Image Processing Sensors," Integrated Design and Process Technology (IDPT), University Jena, June 2007.

[10] http://en.wikipedia.org/wiki/OpenRISC

[11] Damjan Lampret, "OpenRISC 1200 IP Core Specification," September 6th 2001

[12] Xizhen XU and Sotirios G. Ziavras, " A Configurable and Scalable SIMD Machine for Computation-Intensive Applications," Department of Electrical and Computer Engineering New Jersey Institute of Technology Newark, NJ 07102 USA.