Reconfigurable Communication Networks in a Parametric SIMD Parallel System on Chip

Mouna Baklouti, Philippe Marquet, Jean Luc Dekeyser, and Mohamed Abid

INRIA, University of Lille, France
CES, University of Sfax, Tunisia
{mouna.baklouti,philippe.marquet,jean-luc.dekeyser}@lifl.fr
{mohamed.abid}@enis.rnu.tn

Abstract. The SIMD parallel systems play a crucial role in the field of intensive signal processing. For most the parallel systems, communication networks are considered as one of the challenges facing researchers. This work describes the FPGA implementation of two reconfigurable and flexible communication networks integrated into mppSoC. An mppSoC system is an SIMD massively parallel processing System on Chip designed for data-parallel applications. Its most distinguished features are its parameterization and the reconfigurability of its interconnection networks. This reconfigurability allows to establish one configuration with a network topology well mapped to the algorithm communication graph so that higher efficiency can be achieved. Experimental results for mppSoC with different communication configurations demonstrate the performance of the used reconfigurable networks and the effectiveness of algorithm mapping through reconfiguration.

Key words: Reconfigurable architectures, communication networks, SIMD processors, parallel architectures, FPGA

1 Introduction

Embedded image or signal processing applications require high performance systems and highly integrated implementation solutions. They are mostly developed on embedded systems with high performance processing units like DSP or Single Instruction Multiple Data (SIMD) processors. While SIMD systems may have been out of fashion in the 1990s, they are now developed to make effective use of the millions of transistors available and to be based on the new design methodologies such as IP (Intellectual Property) reuse. Nowadays we have a great variety of high capacity programmable chips, also called reconfigurable devices (FPGAs) where we can easily integrate complete SoCs architectures for many different applications. Due to the inherent flexibility of these devices, designers are able to quickly develop and test several hardware(HW)/software(SW) architectures. In this paper, we used Altera [16] reconfigurable devices to implement the mppSoC (massively parallel processing System on Chip) architecture and get experimental results. Our contribution to SIMD on-chip design domain

consists in the implementation at an RTL abstraction level of a parameterized system with flexible reconfigurable networks: one dedicated to neighboring communications and one to assure point to point communications. Reconfiguration is accomplished through instructions. The designer can choose the appropriate mppSoC configuration to execute a given application.

This paper is structured as follows. Section 2 presents several SIMD architectures and focuses on the implementation of their interconnection networks. Section 3 briefly introduces the mppSoC platform. Section 4 details the integration of reconfigurable communication networks in the mppSoC design. Section 5 discusses some algorithms varying the used interconnection network. Finally, Section 6 summarizes the contribution with a brief outlook on future work.

2 Related Works

Due to rapid advancement in VLSI technology, it has become feasible to construct massively parallel systems, most of them are based on static interconnection structures as meshes, trees and hypercubes. Typically, an SIMD implementation [3] consists of a Control Unit, a number of processing elements (PE) communicating through an interconnection network (ICN), which often are custom-made for the type of application it is intended for. If the ICN does not provide direct connection between a given pair of processors, then this pair can exchange data via an intermediate processor. The ILLIAC IV [5] used such an interconnection scheme. The ICN in the ILLIAC IV allowed each PE to communicate directly with 4 neighboring PEs in an 8x8 matrix pattern. So, to move data between two PEs, that are not directly connected, the data must be passed through intermediary PEs by executing a programmed sequence of data transfers [4]. This can lead to excessive execution time if more irregular communications are needed. The same problem of communication bottleneck is encountered with other architectures like [10] and [13], which may cause a significant increase in the cycle count of the program. Other new SIMD architectures have been proposed [7] [14] but they don't perform irregular communications since they integrate only a neighborhood ICN. A nearest neighbor ICN is good for applications where the communications are restricted to neighboring PEs. However, there are several problems which require non local communications. Some massively parallel machines [15] [8] have a scheme to cover such communication patterns. But, they integrate a static ICN. The problem is that different applications might have different demands for the architecture. Several reconfigurable SIMD architectures have appeared. The Morphosys [12] proposed dynamically reconfigurable SoC architecture. It contains a sophisticated programmable tri-level ICN. This gives efficient regular applications, but unfortunately non neighbours communications seem to be tedious and time consuming. RC-SIMD [2] is a reconfigurable SIMD architecture based on two segmented unidirectional communication busses. It is powerful in term of neighboring communications even between distant-PEs, however not efficient for irregular communications. A dynamically reconfigurable SIMD processor array is also described in [1]. It includes a two-dimensional array

of 64x64 identical PEs. It is dedicated to compute programmable mask-based image processing with only support of small local neighboring operations.

Previous proposals appear incomplete from an application perspective. While some architectures are powerful in term of inter-PE communication and some of them are reconfigurable, they can not perform non local operations efficiently. These parallel architectures are not flexible nor scalable to support the requirements of different data parallel applications. The proposed system extends these works by using flexible reconfigurable communication networks based on parametric architecture. In the following, we propose a model of a parallel SIMD system for SoC named mppSoC and we describe its different components.

3 MppSoC Design

MppSoC is an SIMD massively parallel processing System on Chip built within nowadays processors. It is composed of a number of 32-bit PEs, each one attached to a local memory and potentially connected to its neighbours via a regular network. Furthermore, a massively parallel Network on Chip, mpNoC, is able to perform irregular communications. The whole system is controlled synchronously by an Array Controller Unit (ACU). The ACU and PEs are built from the same processor IP (the miniMIPS [17] in this work). The ACU is a complete processor, having 5 stages of pipelining, whereas the PE is a reduced one having only the 3 last execution units. This processor building methodology has a significant gain allowing the integration of a large number of PE on a chip. The ACU transfers parallel arithmetic and data processing instructions to the processor array, and handles any control flow or serial computation that cannot be parallelized. The overall structure of the mppSoC architecture and pipeline is shown in Fig. 1. The pipelined mppSoC architecture has been described in previous papers [11]. Since mppSoC is designed as a parametric architecture, the designer has to set some parameters in order to generate one configuration on FPGA such as the number of PEs, the memory size and the topology of the neighborhood network if it exists.

The mppSoC system is programmed by a single instruction stream partitioned into sequential and parallel instructions. The mppSoC instruction set is derived from the IP processor instruction set used in the design which is modified by adding parallel instructions. Some specific instructions control the two networks, allowing data transfer. Below, we will detail the mppSoC networks.

4 MppSoC Communication Networks

In order to improve the parallel system performances, and to satisfy the requirements of different data parallel applications we propose flexible and reconfigurable communication networks. Availability of such communication is critical to achieve high performance. MppSoC networks are partitioned into two types: regular and irregular networks. The designer can use none, one or both routers to construct the needed mppSoC configuration.

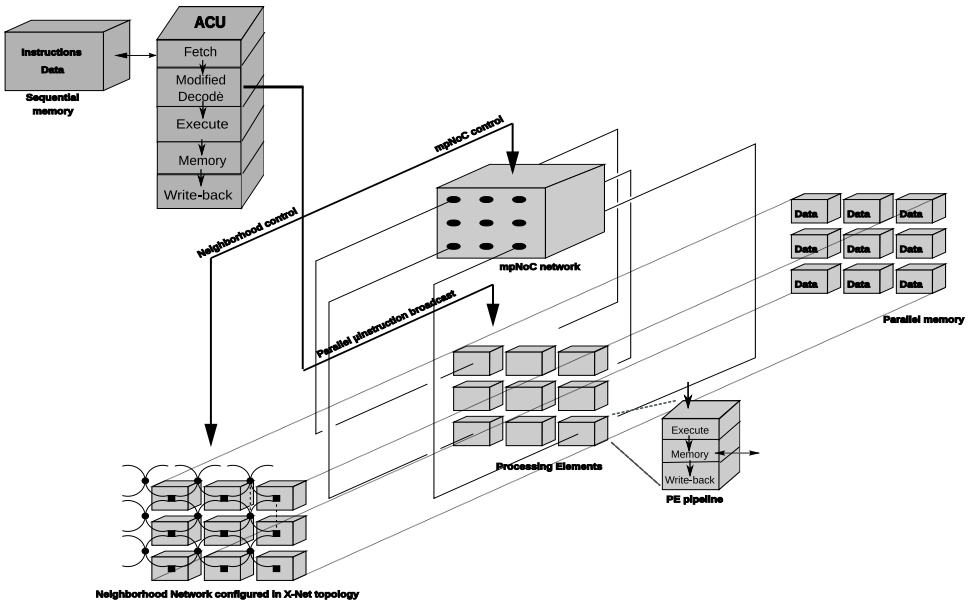

Fig. 1. MppSoC Design

4.1 Reconfigurable Massively Parallel Network on Chip

The mpNoC IP is an irregular network performing point to point communications. It accomplishes three main functions in the mppSoC system. Firstly, the mpNoC is able to connect, in parallel, any PE with another one. Secondly, the mpNoC could connect the PEs to the mppSoC devices. Thirdly, it is able to connect the ACU to any PE. The mpNoC allows parallel I/O transfers solving the need of a high bandwidth required by data parallel applications. It consists mainly of a Mode Manager responsible of establishing the needed communication mode and an interconnection network assuring data transfer. MpNoC input and output ports are connected to switches controlled by the ACU, as shown in Fig. 2. Theses switches allow to connect either the PEs or the I/O devices and the ACU to the mpNoC, depending on the chosen communication mode issued from the ACU to the Mode Manager. In fact, the communication mode could be set at runtime through a mode instruction, when executed the corresponding connections are activated. The mpNoC reconfiguration is performed in two levels. The first level is at compile time where the designer chooses to integrate mpNoC with a selected interconnection network. The second level is during real-time where the communication protocol between the different devices can be altered based on the mode instruction. The proposed mpNoC is scalable according to the number of PEs connected to the network. It integrates an interconnect, responsible of transferring data from sources to destinations, which may be of different types (bus, crossbar, multi-stages, etc). Different networks are provided in a library needed when designing mppSoC. Allowing the designer

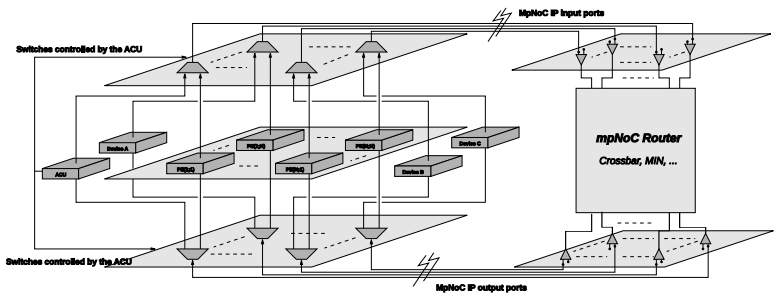

Fig. 2. mpNoC integration into mppSoC

to choose the internal network increases run-time performances. The interconnect interface is generic in order to support a configurable size (4x4, 32x32 for example). While targeting an mpNoC integration into mppSoC, the number of mpNoC sources and destinations is equal to the number of PEs. When using mpNoC, we integrate also a controller to ensure synchronization between PEs since it is the most SIMD important feature. The mpNoC controller verifies if data transferred by the sender is received by the corresponding receiver. At the end of the transmission, the mpNoC controller sends an acknowledgment to the ACU in order to continue executing instructions. The ACU does not issue a new instruction until the communication occurs.

4.2 Reconfigurable Neighbourhood Network

In most data parallel applications, processors work on neighboring data and need to communicate fast among themselves for high performance. Thus a neighbourhood network is also integrated in the mppSoC system. Most common data parallel algorithms need a broad range of processor connectivities for efficient execution. Each of these connectivities may perform well for some tasks and badly for others. Therefore, using a network with a selective broadcast capability, various configurations can be achieved, and consequently, optimal performance can be achieved. We propose different regular network topologies: linear array, ring, mesh, torus, and xnet (a two dimensional toroidal with extra diagonal links). To change from one topology to another the programmer has to use the mode instruction with the appropriate topology value in order to assure the appropriate connections. Five values are defined to specify the provided 5 topologies. In fact, if selected, the neighborhood network with a given topology is generated at compile time. Then the different neighboring links could be changed at run-time. To achieve a high reusability and reconfigurability, the neighborhood network consists of routing elements or switches that are connected to the PEs. Their interface is equipped with 9 ports or interfaces: north, east, south, west, north east, north west, south east, south west and local. The local one is the port that communicates to its attached PE. The switcher activates the appropriate port to transfer data to the needed destination. The way it forwards the data depends on

the executed communication instruction. The network is controlled by the ACU through mode instruction. At every mode instruction, the switches determine a new network topology for the system. In a sense, this is an extension of the SIMD paradigm because for each instruction, the data manipulating the connectivity are controlled in exactly the same way as the data for computing. Circuit switching was adopted to establish the connection, and as a result, a very long path can be established in a large system. In the regular communication, we can specify the distance between PEs on the same row or column or diagonal (in the case of Xnet). The distance defines the number of paths needed to achieve the communication between the PE sender and the other receiver. Consequently, one PE can communicate, not only to his direct neighbour, but also to more distant PE. The nearest neighbourhood network is different from the mpNoC, since it is faster with a less significant communication overhead. In this case, all PE communications take place in the same direction at the same time. Since each interconnection function is a bijection, this transfer of data occurs without conflicts. Sending and receiving data through networks are managed by different communication instructions that will be described in the following subsection.

4.3 Communication Instruction Set

We identify different instructions to program an mppSoC system: processor instructions, micro instructions and specific instructions which are encoded from the processor instructions. Communication instructions, MODE, SEND and RECEIVE, are examples of specific ones. They may be used in different ways to ensure various functions.

MODE instruction serves to establish the needed communication mode in the case of mpNoC or the network topology in the case of neighborhood communication. It relies on the store SW instruction: SW cst, @ModeManager, where:

- @ModeManager = "0x00009003" for mpNoC and "0x00009004" for the neighborhood network.
- cst is the chosen defined value that corresponds to the mpNoC communication mode or the topology of the neighborhood network.

The mode values are defined in the mppSoC configuration file. After setting the required interconnection, data transfers will occur through SEND and RECEIVE instructions.

SEND instruction serves to send data from the sender to the corresponding receiver, based on the SW instruction: SW data, address. The 32bits address can be partitioned in different fields depending on the established mpNoC mode. It contains in case of:

- PE-PE Mode: the identity of the PE sender, the identity of the PE receiver and the PE memory address;
- PE-ACU Mode: the identity of the PE sender and the ACU memory address;
- ACU-PE Mode: the identity of the PE receiver and the PE memory address;
- PE-Device Mode: the identity of the PE sender and the device address;

- ACU-Device Mode: the device address;
- Device-PE Mode: the PE memory address;
- Device-ACU Mode: the ACU memory address.

In the case of regular communication, address contains the distance, the direction and the memory address. There are eight constant direction values, defined in the mppSoC configuration file, that the programmer can specify to denote the eight possible router directions.

RECEIVE instruction serves to obtain the received data, relying on the load memory instruction: LW data, address. It analogously takes the same address field as SEND instruction.

According to his application, the programmer can use all instruction types to satisfy his needs. In the next section, we will present experimental results to validate the proposed mppSoC design.

5 Experiments

In this work, we have executed 3 algorithms: Matrix Multiplication (MM), reduction and picture rotation algorithms, written in MIPS assembly code. In fact, the GNU MIPS assembler has been modified to generate a binary which can be directly integrated in the bit stream of the FPGA mppSoC implementation. The assembly code can be then executed by mppSoC. Each configuration, operating at 50 MHz frequency, is generated in VHDL code and prototyped on the Altera Stratix 2S180 FPGA with 179k logic elements. The proposed system can be efficiently implemented also in any other FPGA family. Different interconnection networks are evaluated and compared with the implemented algorithms. Experimental results were obtained using the ModelSim Altera simulator to simulate and debug the implemented design and the Quartus II which is a synthesis and implementation tool [16] used also to download the compiled program file onto the chip.

5.1 Matrix Multiplication

One of the basic computational kernels in many data parallel codes is the multiplication of two matrices (C=AxB). For this application we have implemented a 64PE mppSoC with mpNoC using two ICN fixed at compile time: a shared bus and a crossbar. As the space of the FPGA HW is limited, 64 is the highest number of PEs that we could integrate on the Stratix 2S180 when integrating the two mppSoC networks. They are arranged in 8x8 grid. The matrices A and B are of size 128x128, partitioned into 8 submatrices A(i,j) and B(i,j), each of size 16x16. Each PE is attached to a 1 Kbyte local data memory. To perform multiplication, all-to-all row and column broadcasts are performed by the PEs. The following code for PE(i,j) is executed by all PEs simultaneously:

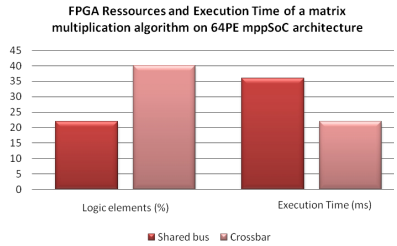

Fig. 3. Experimental results of running a MM algorithm on 64-PE mppSoC

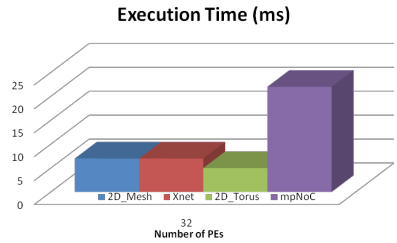

Fig. 4. Execution time results using different communication networks

```
for k=1 to 7 do
send A(i,j) to PE(i,(j+k) mod 8)  /* East and West data transfer */
for k=1 to 7 do
send B(i,j) to PE((i+k) mod 8,j)  /* North and South data transfer */
for k=0 to 7 do
C(i,j)=C(i,j)+A(i,k)*B(k,j)  /* Local multiplication of submatrices */
```

Fig. 3 depicts the FPGA resource occupation and the execution time results. We validated that the architecture based on the crossbar interconnect IP is more efficient but occupies a large area on the chip. In fact, the full crossbar has the particularity to perform all permutations. However, its space on a chip is quadratic depending on the number of inputs and outputs. On the other hand, busses are relatively simple and the HW cost is small. However, we see that the execution time when using a bus is over two times higher than when using a crossbar. This is due to the fact that in a single bus architecture, one interconnection path is shared by all the connected PEs so that only one PE can transmit at a time. We have also tested the use of the neighborhood network (2D mesh selected at compile time) compared to mpNoC. In the mppSoC program we use the mode instruction with the needed topology value in order to change from one topology to another. Fig. 4 shows a comparison between 4 different ICN. As expected, the architecture based on regular network is the most effective for MM application. These tests show also that the torus network is the most appropriate neighbourhood network.

5.2 Reduction Algorithm

The reduction algorithm [8] presents one basic image processing operations. When reduction computation is conducted in parallel, it is known that the computation can be completed with the minimum number of steps using a binary-tree representation as shown in Fig. 5. To implement the reduction algorithm we

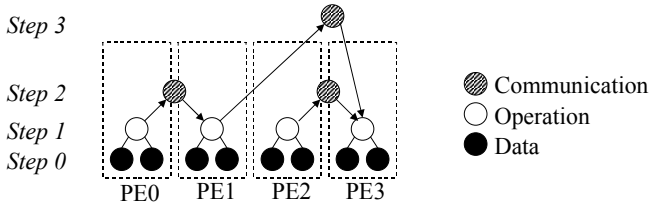

Fig. 5. Parallel reduction computation using four processing elements.

use the recursive doubling procedure, sometimes also called tree summing. This algorithm combines a set of operands distributed across PEs [9]. Consider the example of finding the sum of M numbers. Sequentially, this requires one load and M-1 additions, or approximately M additions. However, if these M numbers are distributed across N = M PEs, the parallel summing procedure requires $log_2(N)$ transfer-add steps, where a transfer-add is composed of the transfer of a partial sum to the PE and the addition of that partial sum to the PE's local sum. The described algorithm (sum of 16384 integers) is executed on mppSoC configurations with 64 PEs (2D and linear) and with topologically distinct interconnection networks (mesh/array neighbourhood network and a crossbar based mpNoC). Execution performances are then compared (Fig. 6). We note

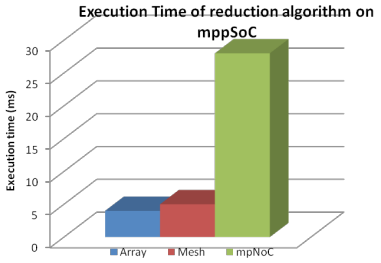

Fig. 6. Execution time on different mppSoC configurations.

that the architecture of the used parallel system has also a great impact on the speedup of a given algorithm. We notice that the mppSoC based on the regular network is the most effective for this type of application. In the case of a

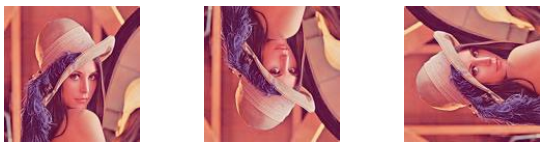

Fig. 7. Picture rotation

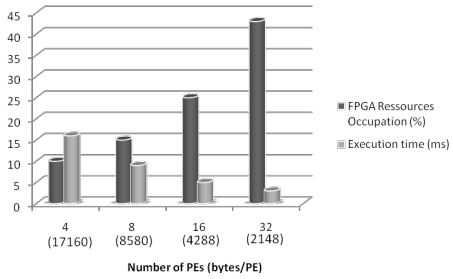

Fig. 8. System scalability/performance running a picture rotation algorithm on a Stratix 2S180

completely-connected topology, the speedup is six times lower than when using a mesh inter-PE network. The two regular topologies, mesh as well as linear array, give approximately the same execution time. Indeed, the time obtained with a linear router is slightly lower than with a mesh router. This is due to the additional communication overhead introduced by the mesh router. So, the linear neighborhood network is the most effective of the reduction algorithm. These different results show also the flexibility of the mppSoC architecture and the high efficiency achieved by establishing a well mapped network topology to one algorithm.

5.3 Picture Rotation

In this algorithm, we realize 17161-pixel picture rotations. The resulting Lena pictures of Fig. 7 were provided by an execution of binary programs on our mppSoC FPGA implementation. The image rotation requires a non homogeneous data movement and I/O throughput requirements. That's why, we have used a crossbar based mpNoC to assure communications and to perform parallel I/O for reading and displaying the resultant image on a VGA screen. So the interconned network in this case is the mpNoC. Selective broadcast capability in this network is enabled by the mode instruction used in the program. We have also tested different number of PEs with variable memory size. Fig. 8 presents synthesis and execution results on various mppSoC designs. We notice a compromise between area and execution time. The results prove the performance of the proposed design. Indeed, when increasing the number of PEs (multiplying by 8) the speedup increases (5 times higher) and the FPGA area is multiplied by a factor of 4 which is an acceptable rate. The results show that the FPGA based

implementation is inexpensive and can easily be reconfigured as new variations on the algorithm are developed.

From all previous experiments we demonstrate the effectiveness of reconfigurable and parametrical networks in a massively parallel system. These networks can perform neighboring as well as irregular communications to satisfy a wide range of algorithm communication graphs. The designer has to make the right choice between the two networks, depending on the application, in order to optimize the whole system performances. In fact, the flexibility and configurability of the mppSoC architecture, in particular its interconnection networks, allow the designer to generate the most appropriate architecture satisfying his needs. It is vital to have a flexible interconnection scheme that can be applied to the system design. The parameterization of the mppSoC is also a key aspect to easily tailor the architecture according to HW as well as SW requirements. The performances of the described system is found better than other architectures. Compared to the ASC processor [10] for example, our mppSoC achieves higher performances (40.48 Mhz with 64 PEs compared to 26.9 Mhz with 50 PEs in the ASC). The mppSoC PEs are 32bits instead of 8bits and contain 3 pipeline stages instead of 4. Compared to the SIMD architecture described in [13], mppSoC is more powerful since it includes a reconfigurable neighboring network rather than a static 2D torus network and it can respond to the irregular communications. In [13], data transfers are based on a global bus which may cause some excessive time since the bus has a limited bandwidth. In term of speed, compared to the H-SIMD architecture [6] mppSoC shows powerful results. In fact, for matrices of size less than 512, the H-SIMD machine is not fully exploited and does not sustain high performance. However, mppSoC is parametric and can be fitted in small as well as large quantity in one FPGA. With a matrix size of 200 the H-SIMD makes 7ms to achieve the computation compared to 5ms obtained when executing multiplication of matrices of size 128 on mppSoC. This comparison prove the high performance and the efficiency of mppSoC.

6 Conclusion

This paper presents a configurable SIMD massively parallel processing system prototyped on FPGA devices. MppSoC can be parameterized to contain several PEs with variable memory size. It is characterized by its reconfigurable communication networks: a neighborhood network and an mpNoC dedicated to irregular communications. Including or not an mpNoC in a given mppSoC design is a trade-off between the cost in term of silicon and the advantage in term of performance and flexibility. To evaluate the mppSoC system we have implemented different sized architectures with various configurations for three representative algorithms. The flexibility of the architecture allows to match the design with the application and to improve the performances and satisfy its requirements. Future work deal with the choice of the processor IP. The ACU for example is not reconfigurable in itself but can be replaceable by an equivalent soft processor or a self designed ACU. The PE could be also obtained by reducing the ACU.

Our aim is to test other processors with the mppSoC design and assure their reconfigurability. The ultimate goal is to develop a complete tool chain facilitating the mppSoC implementation. The nature of the targeted applications may be the decisive element in the design choice.

References

1. D. Ginhac, J. Dubois, M. Paindavoine, B. Heyrman: An SIMD Programmable Vision Chip with High-Speed Focal Plane Image Processing. Eurasip J. on Embedded Systems, Hindawi Publishing Corporation, vol. 2008 (2008)
2. H. Fatemi, B. Mesman, H. Corporaal, T. Basten, R. Kleihorst: RC-SIMD: Reconfigurable communication SIMD architecture for image processing applications. J. Embedded Computing, vol. 2, 167–179 (2006)
3. M. J. Flynn: Some computer organizations and their effectiveness. IEEE Trans. Comput., vol. 21, 948–960 (1972)
4. B. Parhami: Introduction to Parallel Processing: Algorithms and Architectures. (Kluwer Academic Publishers) (1999)
5. R. Michael Hord: The Illiac IV the first supercomputer. (Computer Science Press) (1982)
6. X. Xu, S. G. Ziavras, T. G. Chang: An FPGA-Based Parallel Accelerator for Matrix Multiplications in the Newton-Raphson Method. Lecture Notes in Computer Science, Embedded and Ubiquitous Computing, pp. 458–468. Springer, Berlin (2005)
7. F. Schurz, D. Fey: A programmable parallel processor architecture in FPGAs for image processing sensors. In: Proc. IDPT'07 (2007)
8. H. J. Siege1, L. Wang, J. E. So, M. Maheswaran: Data parallel algorithms. ECE Technical Reports (1994)
9. H. Stone: Parallel computers. In: Introduction to Computer Architecture. H. Stone, Ed. Chicago, pp. 327–355 (1975)
10. H. Wang, R.A. Walker: Implementing a Scalable ASC Processor. In: Proc. International Symposium on Parallel and Distributed Processing, IPDPS, IEEE Computer Society (2003)
11. M. Baklouti, P. Marquet, M. Abid, J. L.Dekeyser: A design and an implementation of a parallel based SIMD architecture for SoC on FPGA. In: Proc. DASIP, Bruxelles, Belgium (2008)
12. M.-H. Lee, H. Singh, G. Lu, N. Bagherzadeh, F. J. Kurdahi, E. M. C. Filho, V. C. Alves: Design and Implementation of the MorphoSys Reconfigurable Computing Processor. VLSI Signal Processing, pp. 147–164 (2000)
13. P. Kumar: An FPGA Based SIMD Architecture Implemented with 2D Systolic Architecture for Image Processing. Available at SSRN, `http://ssrn.com/abstract=944733` (2006)
14. S. E. Eklund: A Massively Parallel Architecture for Linear Machine Code Genetic Programming. In: Proc. International Conference on Evolvable Systems: From Biology to Hardware, pp. 216–224 (2001)
15. T. Blank: The MasPar MP-1 architecture. In: Proc. IEEE Compcon Spring90. IEEE Society Press, San Francisco, pp. 20–24 (1990)
16. Altera, `http://www.altera.com`
17. OpenCores, miniMIPS overview, `http://www.opencores.org/projects.cgi/web/minimips`