

Massively Parallel Processing on a Chip

Philippe Marquet

Samy Meftali

Simon Duquennoy

Jean-Luc Dekeyser

Sébastien Le Beux

LIFL and INRIA-Futurs
University of Lille
France

ABSTRACT

MppSoC is a SIMD architecture composed of a grid of processors and memories connected by a X-Net neighbourhood network and a general purpose global router. MppSoC is an evolution of the famous massively parallel systems proposed at the end of the eighties. We claim that today such a machine may be integrated in a single chip. On one side, new design methodologies such as IP reuse and, on the other side, the possible high level of integration on a chip let us envisage such a revival.

Some improvements of the system architecture are possible because of the high degree of integration: The mppSoC processing elements share most of their design with the control processor, the integrated network allows to exchange data between PEs, but also between the control processor and the PE memories, and even to connect the external devices to the system.

This paper presents the mppSoC architecture, a cycle-accurate bit-accurate SystemC simulator of this architecture, and a prototype of implementation on FPGA. A complete tool chain and the execution of some applications on the simulator and the FPGA implementation validate the modeling choices and show the effectiveness of this design.

Categories and Subject Descriptors

C.1.2 [Computer Systems Organization]: Processor Architectures—*Multiple Data Stream Architectures (Multiprocessors)*; C.0 [Computer Systems Organization]: General—*Modeling of computer architecture*

General Terms

Design

Keywords

Massively Parallelism, SIMD, SoC, System-on-a-Chip, mpp-SoC, MasPar

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CF'07, May 7–9, 2007, Ischia, Italy.

Copyright 2007 ACM 978-1-59593-683-7/07/0005 ...\$5.00.

1. SIMD MACHINE ON A CHIP

By the end of the eighties, massively parallel SIMD computers were much known in the community as high performance machines, especially in terms of computing speed. Those SIMD machines were also known as processor-array machines; they basically consists of an array of fine-grained computational units connected together by different types of simple network topology. The computing power of the machine comes from its relative great number of processors, not from a huge computational power or a high clock frequency of the processors.

Typically, a SIMD machine is a grid of processors; each processor executing the very same instruction at each cycle in a synchronized manner, orchestrated by an unique control processor. This control processor is responsible for fetching and interpreting instructions. Two kind of instructions are considered: parallel ones and non-parallel (i.e. sequential) ones. The control processor transfers parallel arithmetic and data processing instructions to the processor array, and handles any control flow or serial computation that cannot be parallelized.

This class of processors has been then less and less used because of its dramatically high fabrication cost. In fact, it was more interesting for the vendors to use complex standard processors than several specifics ones when designing a new system. A second and technological aspect had also lead to the disappearing of this kind of systems: the operating frequency increase of most of the electronic components had make impossible the broadcast of an instruction to a thousand of elementary processors in a synchronous manner at each clock cycle.

Nevertheless, nowadays, many modern application domains are concerned by the conjunction of regular parallel algorithms and high computing resources. They include signal and image processing applications such as software radio receiver, sonar beam forming, or image encoding/decoding. Furthermore, the integration of the system on a single chip will be of prime interest for those applications that also require some degree of embeddedness.

When targeting embedded computing, one often considers power consumption. Parallelism is known to be an effective answer to the reduction of power consumption, especially when compared to frequency increase: the power consumption is proportional to the circuit size but exponentially increasing with the frequency.

Present system design methodologies promote component based design and enforce the use of IP blocks and industry standard interfaces [10]. This context substantially allevi-

ates the design cost of a dedicated processor for an SIMD system. On the integration side, one can imagine the integration of a full SIMD grid on a single chip, facilitating the synchronous broadcast of an instruction at a high frequency.

Fifteen years after the decline of traditional massively parallel systems, significant evolutions of system design, silicon integration technology and growing application computing power requirement have change the context and it clearly seems important to consider and verify the feasibility and performances of massively parallel machines on a chip.

Contributions and Paper Organization

The objective of this paper is to reconsider the interest and the feasibility of a SIMD machine, especially in the context of a single chip system integration. We propose a novel SIMD machine built within nowadays processors. Our machine is named mppSoC which stands for Massively Parallel Processing System on Chip. Two versions of mppSoC are provided: a SystemC version that allows fast simulation and performance evaluation, and a RTL version that leads to a physical realization onto FPGA. Both realizations are programmable with the same tool flow.

The next section presents some significant works related to SIMD systems and their integration on a chip. Section 3 introduces the mppSoC architecture. A more detailed design and its SystemC implementation are presented in the Section 4. Section 5 presents and discusses the FPGA implementation of mppSoC. The mppSoC tool chain and the execution of some representative applications are briefly described in Sections 6 and 7.

2. RELATED WORKS

The history of SIMD machines began with the ILLIAC IV project [5, 9] started in 1962. The machine was the first large-scale multiprocessor, composed of 64 processors. Danny Hillis resurrected the SIMD architecture in 1985 with his Connection Machine composed of 65,536 1-bit processors [8, 11]. However, following a short stint in the eighties by several commercial companies such as Thinking Machines, MasPar, and Wavetracer, SIMD has once again fallen by the wayside in the arena of commercial general-purpose computing.

Even though SIMD machines have not found their way into large scale development in the end of nineteenth, they have not completely died out. The reason is that SIMD architectures still make a lot of sense for special applications that require a great deal of independent data computation. In addition, multimedia and graphics applications have become very common place with the advent of the internet and computer-gaming. Since multimedia such as video or sound and signal processing are inherently parallelizable tasks that involve computation on data streams, small-scale commercial SIMD variants have come into the marketplace. Thus, SIMD has once again found a way to survive in the form of ISA multimedia extensions, including Intel's MMX and SSE, AMD's 3D-Now, and Motorola's AltiVec.

With the recent great evolution of silicon integration technology, some major IP providers start designing special components for on-chip SIMD architectures acceleration. For instance, the Neon technology proposed by ARM [2] is an architecture option of the with ARMv7A architecture that provided a 64/128-bits hybrid SIMD architecture to accelerate the performance of multimedia and signal processing ap-

plications. This significant evolution means that it becomes today economically interesting to design SIMD machines with hundreds of processing units and embedded memory on a single chip. Thus, we start seeing in the last years some small sizes SIMD on-chip systems.

MorphoSys [15, 20, 6] is a reconfigurable SIMD architecture that targets for portable devices. It combines an array of 64 Reconfigurable Cells (RCs) and a central RISC processor (TinyRISC) so that applications with a mix of sequential tasks and coarse-grain parallelism, requiring computation intensive work and high throughput can be efficiently implemented on it. In MorphoSys, each RC can communicate directly with its upper, below, left and right neighbors peer to peer. This gives efficient regular applications, but unfortunately non-neighbors communications seems to be tedious and time consuming.

The picoChip's PC101 array [4, 19] integrates more than 400 heterogeneous 16-bit processors on a single die. The processor arrays have enough redundancy to tolerate some defective elements during manufacture. It is a kind of very high performance DSP, furnished with a complete programming environment based on C. It seems to be a strong platform for wireless application, but it doesn't benefit from regularity aspects of many parallel applications.

Unfortunately, to our knowledge, there are no available massively parallel SoCs, with significant number of processing unit, and using really the recent integration technology capabilities.

3. MppSoC ARCHITECTURE

MppSoC is a SIMD machine belonging to the class of processor array machines: MppSoC is composed of a 2-D grid of processors (the PEs) working in a perfect synchronization. A small amount of local and private memory is attached to each PE. Each PE in the 2-D grid is potentially connected to its 8 neighbours via the X-Net, a regular network. Furthermore, each PE is connected to an entry of mpNoC, a massively parallel Network on Chip that potentially connects each PE to one another, performing efficient irregular communications. The system operating is organized by a control processor (the ACU, Array Control Unit) that accesses its own memory. Figure 1 illustrates the global architecture of mppSoC.

The whole mppSoC operating is controlled by the ACU in a synchronous manner. At each cycle, each PE executes the same instruction that have been broadcasted by the ACU. The ACU also controls in the same synchronous manner the two networks of the system: the X-Net and mpNoC.

MppSoC is programmed in a data-parallel language. A data-parallel language distinguishes sequential instructions and parallel instructions. A sequential instruction concerns sequential data of the ACU memory and is carried out by the ACU as in a usual sequential architecture. A parallel instruction is executed in a synchronous manner by all the PE of the system, each PE taking its operands from its local memory and storing the result in this same memory (or maybe in its own local registers). Some specific instructions control the two networks, allowing transfer of values from one PE to another. These transfers are also executed in a synchronous manner: all the PE communicating at the same time with a PE designated by the instruction or one of its operand values.

The whole system is synchronously controlled by the ACU

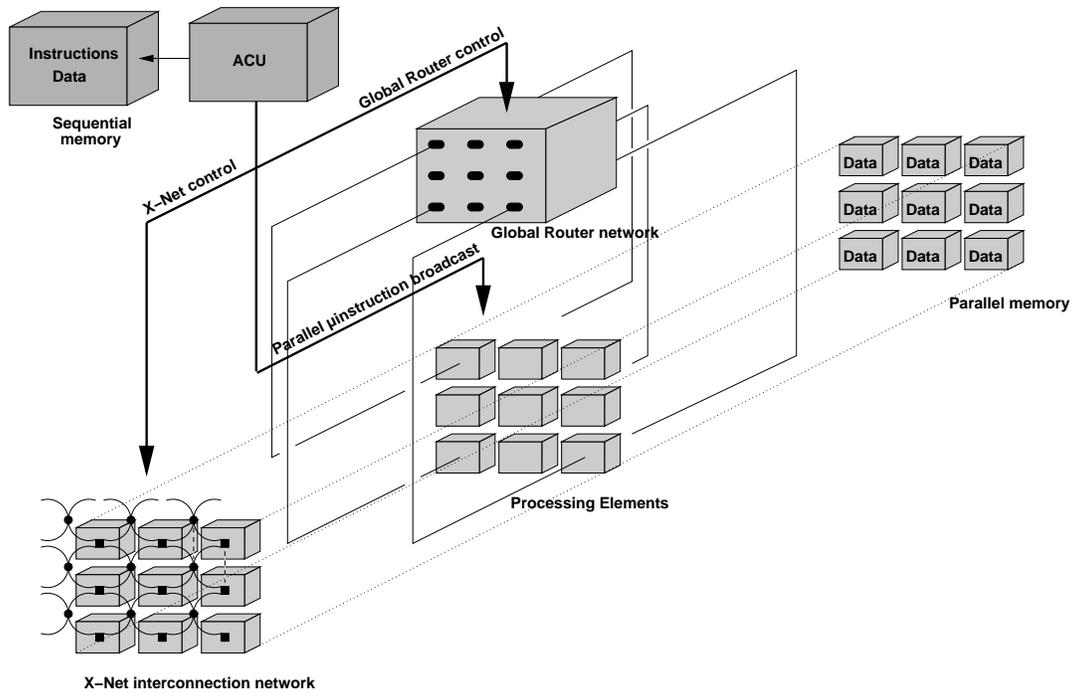


Figure 1: The mppSoC architecture. The ACU is connected to its sequential memory, which contains data and instructions. The ACU reads instructions and, according to the executed program, synchronously controls the whole mppSoC architecture within these three ways: a) parallel instructions which are broadcasted to the 2-D grid of PEs, b) X-Net controls which establish the connection topology of this interconnection network and c) the global router control which determines the route of the irregular communications. Each PE is connected to its local data memory (on the right of the figure), a X-Net node (on the bottom left-hand side) and to the global router (in the center of this figure), which is realized with mpNoC.

that decodes an instruction at each clock cycle. Sequential instructions are directly carried out by the ACU, while parallel instructions are broadcasted to all the PEs in the form of what we call a micro-instruction. Communication instructions concerning one of the two networks are also decoded by the ACU that broadcasts commands to the networks and micro-instructions to the PEs to realized those instructions.

Thus the PEs synchronously receive a micro-instruction. The local execution of the micro-instruction is dependant of the local state of the PE: a PE may be active or not. This activity state is store in the activity bit of each PE. This activity bit is controlled by dedicated instructions (set, and, or, etc.). All the PEs receive the micro-instructions, but only the PE with their activity bit set executes the micro-instructions. The other PEs do nothing.

The design of mppSoC is inspired from the famous MasPar [3, 17]. Nevertheless three major points distinguish mppSoC from the MasPar:

1. The mppSoC PEs are not any more small 1- or 4-bit processors as it was by the time of the Connection Machine CM-1 ar MasPar MP-1. MppSoC uses 32-bit processors.
2. The ACU and the PEs are designed from a same processor. Some minor additions are made to this processor to design the ACU, while its decode part is suppressed in the PE, performing a better on chip integration and reducing the power consumption.

3. The mppSoC global router not only connects the PEs to each others, but also allows to connect the PEs to ACU and to the devices.

Let us detail these three points.

In term of performances, it is clearly a great advantage to use a 32-bit processor rather a small 4-bit parallel ALU and this trend was observed with the second generation of SIMD machine proposed in the nineties such as the MasPar MP-2 [12]. Nevertheless, floating point operations were still taken considerable time on those machines (25 cycles for a floating point add on a MasPar MP-2 while an integer addition takes 3 cycles!). If the integration of a pipelined PE was suggested [1], to our knowledge, no such a machine were realized despite the interest of floating point arithmetic in several applications targeted by parallel architectures.

The second difference between traditional SIMD machines and our mppSoC is the common design of the two processors: the ACU and PE. Because of the specialization of the PE, these processors were usually designed for that sole usage. We propose to decrease the cost of this dedicated design by using a single and preexistent processor as a base for the design of the ACU and the PEs. We consider a typical pipelined RISC processor and identify several stages in the instruction pipeline of the processor: the first stages fetch and decode the instructions while the following ones execute the instructions. The ACU is build as a modified processor which produces a micro-instruction at the end of the decode stage. This micro-instruction is either executed locally by

Table 1: The mpNoC instruction set

r_open PRt	Open a Global Router communication channel. PRt contains the destination PE. Set R to 1 for each destination PE.
r_send PRd,PRs	Send data via the open channels. PEs with R bit set to 1 receive a data. $Dst(PRd) \leftarrow Src(PRs)$
r_osend PRt,PRd,PRs	Open then send data on the router.
r_fetch PRd,PRs	Fetch data from the open channels. PEs with T bit set to 1 can receive a data. $Src(PRd) \leftarrow Dst(PRs)$. Set F bit to 1 for fetching PEs.
r_close	Set T bit to 0 for open channels PEs.
r_fetchc PRd,PRs	Fetch data from the open channels then close channels.
r_mode SRh	Set the mpNoC mode (PEs to PEs, devices to PEs, PEs to devices).

PRx: the PE Rx register

SRx: the ACU, thus scalar, Rx register

R and T bits belong to the register status of each PE

the ACU in the case of a sequential instruction, or is broadcasted to all the PEs in the case of a parallel instruction.

Another major difference between usual SIMD systems and our mppSoC is the integration of mpNoC, Massively Parallel Network on Chip, a multi-purpose NoC component in the mppSoC. MpNoC was designed as an IP that is able to synchronously connect a set of inputs to a set of outputs [7]. The mpNoC IP fulfills a triple function in the mppSoC architecture. Firstly, the mpNoC is used as a global router connecting, in parallel, any PE with another one. Secondly, the mpNoC is able to connect the PEs to the mppSoC devices. Finally, the mppSoC is able to connect the ACU to any PE of the mppSoC. Several implementations of the mpNoC have been proposed (one is based on a full crossbar network, another on an extended delta network [13]). An effective implementation can be chosen depending, for instance, on the number of PEs.

4. MppSoC COMPONENT DESIGN

The mppSoC general architecture was described in the previous section, we will now detail the way the mppSoC SystemC simulator and the RTL implementation are designed. While some IP construction, such as memories, does not require many attention, some others, such as the processors, are much more complex. Communication IP, mpNoC and X-Net, are tedious to integrate due to the number and the heterogeneity of connexion they manage. The following deals with particularities of each IP block and its integration into mppSoC.

4.1 Processors

Both the ACU and the PEs are derived and completed with several functionalities from the MIPS version furnished in SoCLib [21]. The choice of the MIPS processor was mainly based on pragmatic reasons: the MIPS is a quite simple, standard, representative and well established processor. Furthermore the SystemC MIPS provided by SoCLib is well designed and allows easy extensions and modifications.

The MIPS instruction set is extended in two ways:

1. Most of the arithmetic, logical instructions are dupli-

cated to a usual sequential version (executed by the ACU) and a parallel version (executed by each PE). Identically, memory access instructions are duplicated. The sequential memory access instructions allow the ACU to access its memory, while the parallel accesses permit each PE to access its own local memory.

2. Some mppSoC specific instructions are added to the MIPS, mainly:
 - X-Net communication instructions. Typically such an instruction allows each PE to write a local value in the register of another PE, the distance and direction between the two PE are the same for all the PEs. As for all parallel instructions, these instructions are broadcasted to all the PEs, but the ACU also broadcasts commands to the X-Net components to establish the communication links;
 - mpNoC communication instructions. A mpNoC communication is realized in several steps: a channel must be opened before sending data and must be closed after. Table 1 summarizes these instructions, which are broadcasted to the PEs and used to control the mpNoC components. The publication [7] details these instructions and their realization;
 - a set of instructions that manipulates the PE status register and especially allows each PE to set its activity bit. These instructions are broadcasted to all the PEs that locally operate on their status register;
 - a broadcast instruction that move a value from an ACU register to a register of all the PEs.

Each of the added instructions uses one of the instruction format of the original MIPS. We complete the original MIPS instruction table with some particular attention in order to facilitate the instruction decoding.

As already mentioned, each PE has an activity bit that indicates if the PE locally executes the instruction or remains

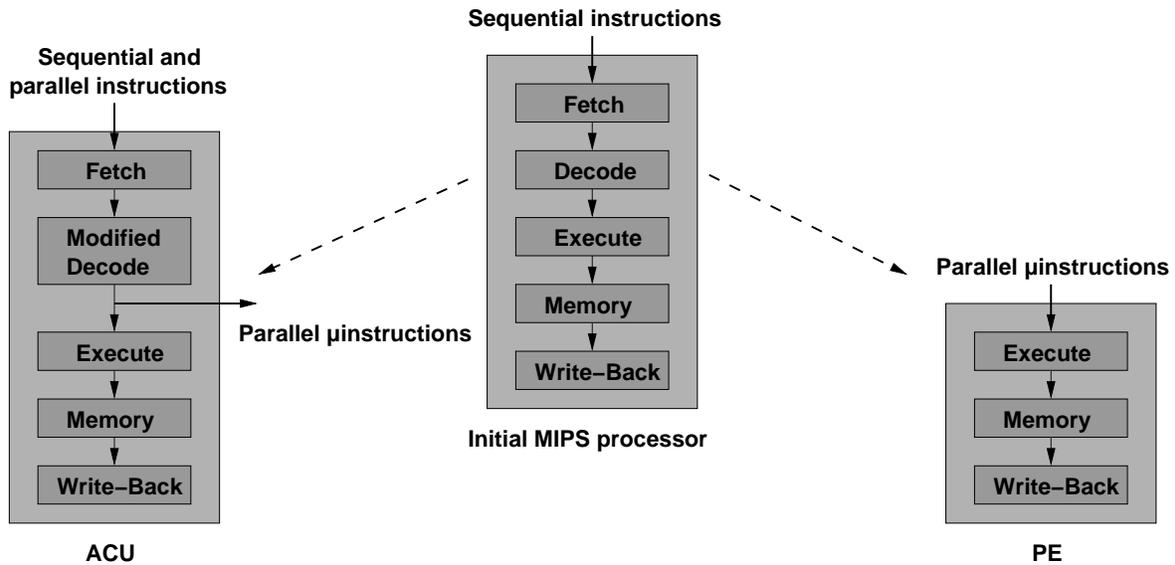


Figure 2: The ACU and PE are based on a the MIPS processor. The chosen initial MIPS processor is shown in the center of the figure. This processor is able to decode sequential instruction. According to the extended instruction set, the decode pipeline stage is modified, creating the ACU. The ACU is able to decode both sequential and parallel instructions. The micro-instruction issued from a decoded parallel instruction is sent out from ACU. Parallel micro-instructions are consumed by PE which is build from the three last pipeline stages of the initial MIPS processor.

idle. Of course the instructions manipulating this activity bit are systematically executed, whether the activity bit is set or not. Each PE also contains another dedicated register that returns its number in the PE grid and that can be used to particular its own job. On the ACU a dedicated register is continuously updated with a global or reduction of the activity bits of the PEs.

The MIPS processor includes a 5-stages pipeline. The firsts stages of the pipeline load and decode the instructions. These stages are only implemented in the ACU. The three last stages of the ACU then execute the instruction per se in the case of a sequential instruction, or send the command to the micro-instruction to the PEs and, if necessary, the command to the communication components. In the first simulator implementation the commands are sent to the PE in the form of the rough decode instruction, as illustrated in the Figure 2. The component that implements the PEs simply contains the last three stages of the MIPS pipeline that directly receive these micro-instructions. Consequently, the PEs have no PC register.

Thus, because they do not include the first decode stages, the PEs are less complex than the ACU. This significant gain will allow to integrate a large number of PE on a chip.

Two memory components are distinguished in mppSoC: the sequential memory attached to the ACU and the memory attached to each PE. The set of these later forming the parallel memory of the system. All the memory components provide a VCI interface (Virtual Component Interface, a norm of the VSIA consortium [22]). Each of these interfaces is connected to a PE or the ACU.

4.2 Neighborhood Network

The mppSoC X-Net network is inspired from the MasPar. It gathers all the PE in a 2-D grid, allowing each PE to

communicate with its eight neighbour PEs, and by extension to any PEs in the eight main directions.

This 8-directions communication is realized via two kinds of components: 5-pins X-Net components and 4-pins X-Net components. Each PE is attached to a 5-pins component, this 5-pins component is connected to four 4-pins components. 5- and 4-pins components are configurable so they can realize the connection between any two of the components they are connected to. The ACU will drives this configuration. The Figure 3 illustrates the X-Net connections.

A given X-Net communication allows all the (active) PEs to communicate with a PE in a given direction at a given distance. Direction and distance are here the same for all the PEs. Such a communication is realized in several communication phases driven by the ACU that sends the appropriate micro-instructions to the PEs and commands to the X-Net components. Thus, the `xnet` instruction has been added to the MIPS instruction set and is only executed by the ACU that sends `OP_WRITE_XNET` and `OP_READ_XNET` micro-instructions to the PEs.

The `xnet_c` instruction is used for some advanced communication schemes requiring some restrictions on the activity of intermediate PEs between any couple of communicating PEs lead to more efficient implementations of the communication. Identically, the `xnet_p` instruction allows each PE to broadcast a value to all the PEs in a given direction, as far as all the intermediate PEs are inactive.

4.3 MpNoC and Global Router

MpNoC is the network component of the mppSoC that allows the parallel communication of each PE to a distinguished PE. MpNoC works using circuit switched: `r_open` and `r_close` instruction configures the mppSoC (see Table 1, for instruction parameters), while `r_send` and `r_fetch` trans-

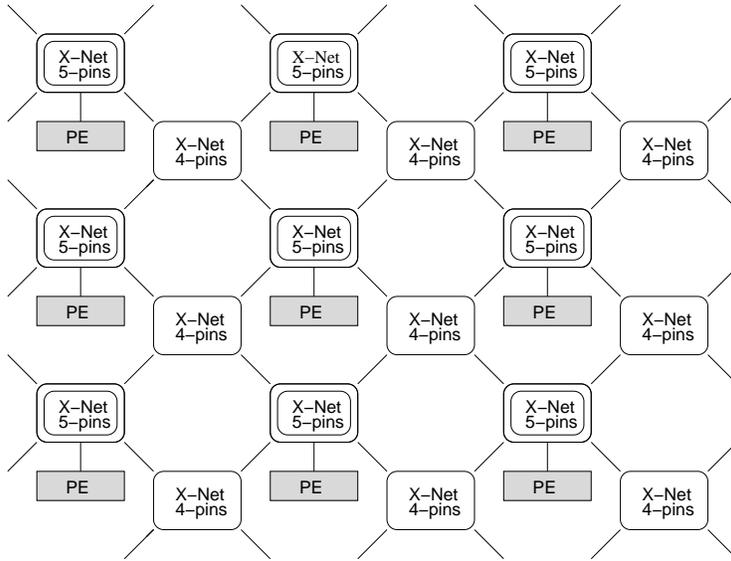


Figure 3: The X-Net network. Each PE is connected to 5-pins X-Net component, represented within a double box. Each 5-pins X-Net component is connected to four 4-pins X-Net components. According to a configuration provided by the ACU and based on the current instruction, the X-Net network allows each PE to be connected to one of its 8 neighbours.

mit data. PEs are not directly connected to the mpNoC but are connected to switched that allows to connect either the PEs, either the ACU, or some devices to the mpNoC. See Figure 4. These switched are controlled via the ACU based on the `r_mode` instruction.

We have several SystemC implementations of the mpNoC. We use a full crossbar for mppSoC systems with a small number of PEs, and a more complex network (EDN, extended delta network) for big instances of mppSoC. Despite its good properties [13], the EDN network is not a full network: a given communication pattern may need several communication phases. These iterations are managed at the mppSoC instruction level using some dedicated state bit of the PEs registers.

4.4 MppSoC SystemC Implementation

The general design of the mppSoC architecture is specified by a SystemC modeling. This implementation of a mppSoC simulator allows clarifying many implementation details of the architecture. All the components are described in SystemC at CABA level (Cycle Accurate Bit Accurate) and have standard VCI interfaces. The whole simulator consists of about 5.000 lines of SystemC source code.

5. MppSoC FPGA IMPLEMENTATION

A first implementation of the mppSoC was set as a feasibility study of the designed architecture. Targeting a FPGA platform was chosen for the sake of its, relative, simplicity and accessibility.

This effective implementation allows to detail the design of the mppSoC architecture. For example, it has been necessary to modify the instruction set encoding to reduce the signals from the ACU to the PEs. In other respects, a accurate design of the instruction pipeline and its coherence between the ACU and PEs has also been elaborated.

This implementation was also a practical experiment of

an IP reuse. Both the ACU and PE implementations were designed from the miniMIPS [18], a preexistent MIPS implementation on FPGA.

Finally, the choice of a FPGA implementation allows a generic design that can be tuned with respect to the effective targeted application. Our prototype generated a FPGA configuration parametrized by the number of PEs and the memory size of each PE. In fact, an application may require many PEs with a short memory, or a small amount of PE with a large memory.

5.1 Implementation Overview

The current implementation of mppSoC on a FPGA includes three main components: an ACU, a given number of PEs, a given amount of local memory attached to each PE, as illustrated on the Figure 5. All the signals between these three components are of course included. The fact the ACU is allowed to access the memories of any PE is used as a palliative to exchange data between ACU and PEs, and even between PEs.

We target the Altera Stratix-2 FPGA that contains 60k logic elements (LEs), a number of dedicated memory blocks, and 36 DSP blocks (36×36 multipliers). Our implementation is a VHDL code that can be used either as an input of the ModelSim simulator [16], or as an input of the Quartus synthesis tools. Both the simulation and synthesis are parametrized by the number of PEs and by the amount of local memory attached to each PE.

The mppSoC board contains an ACU and a grid of PEs. The grid elements are included in a hierarchical unit that contains a PE, its memory, and its chip select. An intermediate hierarchic level grouping several PEs has been defined to help the synthesis tool and to facilitate the routing.

The mppSoC processor is based on the five-stage pipeline miniMIPS processor: fetch, decode, execute, memory, and write-back. The ACU contains the five stages. The decode

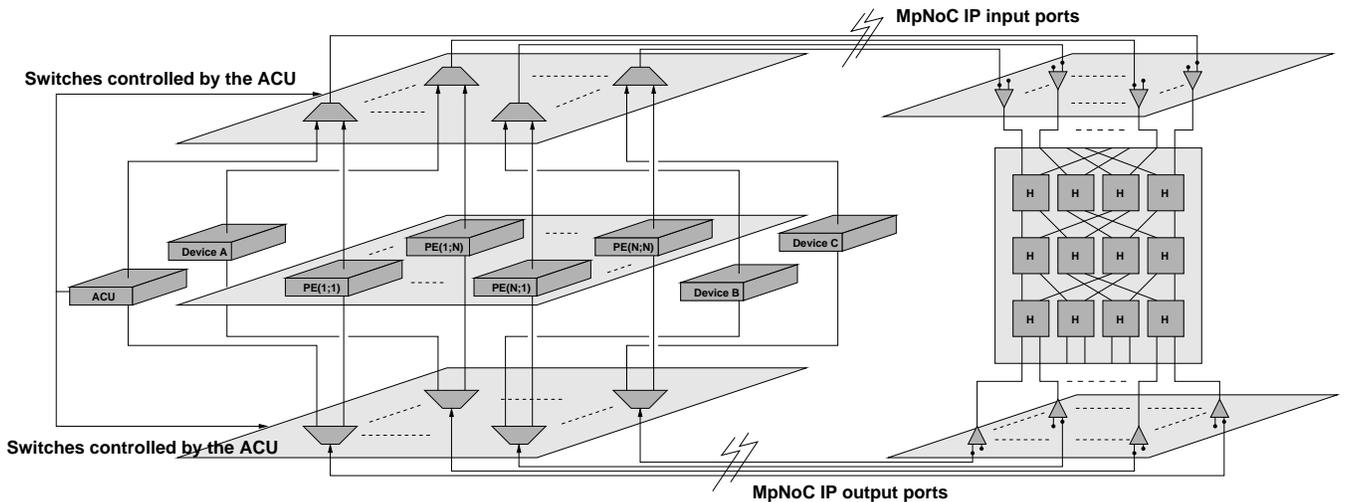


Figure 4: MpNoC integration into mppSoC. The mpNoC and its input/output ports are shown on the right-hand side of the figure. Like one of our implementation, it includes some converters and an EDN. The mpNoC is connected to mppSoC and their input/output devices via controlled switches (top left and bottom left-hand side). The mppSoC (including the PE grid and the ACU) and input/output devices are situated between those switches.

stage has been modified: a vector instruction send the micro-instruction to the PEs and a `nop` in the ACU execute stage, while a sequential instruction makes operations in reverse order. The PE pipeline only contains the three execute, memory and write-back stages. The input from its execute stage is connected to the decode stage of the ACU. The PE write-back stage is controlled by the activity bit of each PE: operations are realized but their results are not store in registers or memory.

The ACU pipeline has also been modified to implement multi-cycle instructions. The only instruction that takes two cycles is the broadcast of a 32 bit ACU register to the PEs via the 16 bits data bus. All the other instructions are realized in one cycle.

The memories are implemented on the dedicated blocks of the FPGA.

The study of the signals delivered from the ACU to the PE has required all our attention. In fact, a large number of signals allows to factorize some work in the ACU and thus to simplify the PEs, while a great number of signals uses a significant amount of place on a board. We have choose to restrict the number of signals without breaking our law that the PE and the ACU are two synchronized variants of the same processor (i.e. they execute the same micro-instruction set). This compromise solution takes care of the necessary area while keeping the design simple.

5.2 Implementation Outcomes

Our mppSoC prototype implementation on FPGA has validated several results. Beyond the feasibility confirmation of our design, we have been able to identify a rough idea of the number of LEs needed to implement a given mppSoC configuration. We have been able to implement a 16 PEs, with 4kB of memory per PE, mppSoC on our 60k LE Stratix-2. The number of LEs has been observed to be linear to the number of PEs. We then interpolate that a 128 PEs should be implemented on the upcoming next generation of

FPGA. Furthermore, our implementation uses not only the LEs, but also the memory and DSP blocks of the FPGA in a well-proportioned manner.

We have also been able to evaluate the maximum frequency of our prototype. Several factors limit this maximum frequency, the first one is the 60 MHz speed of the initial miniMIPS processor. We have been able to reach a 50 MHz frequency for some small configurations of mppSoC. However, a maximum of 42 MHz has been possible with the 16 PEs configuration. The increase of the number of destination of some signals, the growing length of these signals and a more complex place-and-route on an overloaded FPGA explain this reduction. Nevertheless, this 42 MHz frequency is a good result compared to the initial 60 MHz.

The analyse of the FPGA surface utilization shows that a PE is much smaller that the ACU. The decode stage is the biggest stage of the ACU (1350 LEs). The execution stage (950 LEs on the ACU, 1090 on a PE) uses the greatest part of the FPGA LEs.

The analyse of the signal shows that most of them are used to connect the ACU to the PE local memories. Considering that this instruction is not frequently used and that most of the SIMD machines do not have such connection, this double access the PE memory will be reconsidered in our future designs, specially with an increasing number of PEs and not to mention a future global router implementation that will offer such a functionality.

6. MppSoC TOOL CHAIN

As already mentioned, the MIPS instruction set was extended to program the mppSoC. From a mppSoC assembly code, the mppSoC compiler generates a binary that can be used either by the simulator or by the FPGA implementation. This mppSoC compiler is a modification of the GNU MIPS assembler.

The execution of mppSoC binary first needs to load the binary into the instruction memory of the system. The Sys-

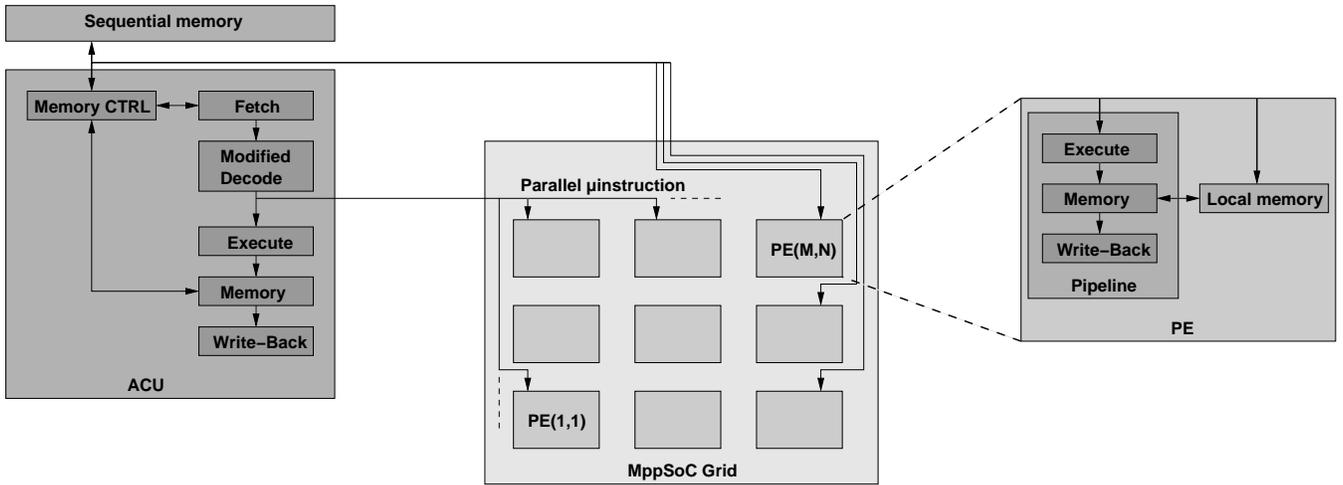


Figure 5: First FPGA implementation of mppSoC. The left-hand side represents the ACU and its sequential memory. The 2-D grid of PEs is represented in the center of the figure. The mppSoC FPGA implementation is different from the SystemC one because it does not yet integrate X-Net and mpNoC. Thus, a bus connecting the ACU to the parallel memory has temporarily been integrated, as suggested by the connection to the PE local memory, on the right-hand side of the figure. This bus permits basic data exchanges in this very first FPGA implementation of mppSoC.

temC simulator has a dedicated function to do so. Concerning the FPGA implementation, the binary program must be integrated in the instruction memory and, after the synthesis, is deployed with the bitstream to the FPGA board.

Furthermore, we developed a set of CPP macros that allows to identify the parallel and sequential variables and expressions. These macros use the `asm` constructs of CPP to make the GNU MIPS compiler generates some marks in the assembler code. This basic method facilitates the manual editing of the MIPS code to produce a mppSoC assembler code.

With these tools, it has been possible to compile and to execute programs on mppSoC. In the following, we present some execution results of various applications.

7. FIRST MppSoC APPLICATIONS

This section briefly presents the very first applications we developed on mppSoC and their results, either on the mppSoC SystemC simulator or on the mppSoC FPGA implementation.

7.1 Mandelbrot Fractal

The Mandelbrot fractal is a representation of a mathematical set of complex numbers. Each point of the plan represents a complex number. A simple formula permits to know if a complex is inside the set. The algorithm is iterative. For a given number, if the number is not in the set, the algorithm stops after a variable number of iterations. If not, it does not stop.

The idea to represent the fractal is to assign a colour to each point according to the number of iterations need to convergence. For instance, if using 256 levels of gray, one stops after 256 iterations. There is then a range indicating the more or less fast convergence of calculation.

We have executed this fractal algorithm onto mppSoC. At the end of the simulation, a memory dump is automatically realized. Within this dump result, we observe that 84%

of the ACU instructions are parallel. This is surely very interesting for an SIMD system. In fact, this means that usually, a major part of the chip is used. We see also that a mean of 34% of the PEs execute an instruction in each cycle. The execution speed is about 35 billions instruction per second, while the theoretical maximum speed is about 100 billion instructions/second.

7.2 Correlation Algorithm

In order to estimate the mppSoC efficiency while targeting an intensive signal processing task, we study an implementation of the following correlation algorithm:

$$S_t = \sum_{i=1}^{1023} (C_i \times Y_{i+t}) \quad (1)$$

S is the algorithm output, C is the reference code correlated with the input signal Y . Since this algorithm is used in an automotive context [14], we define a mppSoC grid size allowing an implementation on the target FPGA and offering powerful enough computation power. We thus use the grid composed of 16 PEs, proposed in Section 5. According to the target automotive application specification, the correlation windows is equal to 1023.

Two main solution allow the algorithm implementation on mppSoC (16 PEs):

- the **coarse grain** implementation shares correlation points to compute between PEs. No data exchange between PEs are required for this solution while considering an initial presence (and thus load) of C and Y in PEs local memories.
- the **fine grain** implementation shares parts of the correlation points to compute between PEs. This solution introduces partial correlation results exchange, but the computation can begin even if C and Y are not stored



Figure 6: Picture rotation

in PEs local memories. The latency is considerably reduced compared to the **coarse grain** implementation, but the computation time itself increases due to the large number of data exchange. Moreover, communication (X-Net or global router) does not yet exist in the current mppSoC implemented on FPGA. For these reasons, we did not retain this solution.

The formula 1 is implemented on 16 PEs sharing computation of the 1023 correlation points which compose one correlation windows. Each PE executes 64 correlation points, the last one (the 1024th) is not retained. The algorithm on Figure 7 illustrates the proposed implementation.

```

dec = 1023 - PEindex ;
for cpt=1 to 64 do
  S = 0;
  for i=1 to 1023 do
    S += Y[i] * C[dec + i mod 1023];
  end
  s += reduceAdd(S);
  dec -= numbersOfPEs;
end

```

Figure 7: Correlation algorithm executed on each PE. The *reduceAdd()* function takes a value on each PE and combines them, by addition, in an ACU value.

Loading the input signal Y into all PEs local memories (by a parallel way) requires $75\mu s$ for a 50Mhz mppSoC runtime frequency. The 16 first correlation points results are produced and returned to the ACU after $408\mu s$. Thereafter, 16 new results are correctly returned after $330\mu s$. The whole correlation windows is obtained after 21ms, which is powerful enough for the proposed case study [14].

7.3 Picture Rotation

Using the mpNoC can be very interesting to realize several kinds of communications. However, the choice of using X-Net or global router in mppSoC platform for a given application is not at all obvious. In fact, using X-Net in non-regular communications is very tedious, while using global router in regular ones is time and energy consuming. Thus, the designer has to make the right choice between the two networks, depending of the application, in order to optimize the whole system's performances.

Picture rotation algorithms seem to be simple and good examples to use the mpNoC. In fact, in this situation, communications are very irregular: PEs need to communicate using several different directions and lengths. Obviously, it is possible to realize that using the X-Net network, but this needs several communication steps (as much as the number of PEs). We tried this rotation using the mpNoC on a picture. With a PE grid of 32×32 PEs, we realized 1024-pixel picture rotations. Using the EDN based mpNoC, only sixteen communication steps were required to realize one picture rotation.

This result shows that using the mpNoC instead of the X-Net can be really efficient in terms of performances and easier to program. The resulting pictures of Figure 6 were provided by the execution of a binary program on our SystemC mppSoC simulator.

8. CONCLUSION

This paper gives a complete view of mppSoC platform. It consists in a configurable regular grid of MIPS processors that communicate via a X-Net network, and mastered by an ACU. In addition to local and global memories, a global router network is furnished with mppSoC. It allows fast non regular communications between processing elements. Our platform is entirely described in SystemC at cycle accurate bit accurate level, in order to allow fast and accurate simulations. We also present a complete programming, compiling and execution set of tools to facilitate the use of mppSoC by application designers. A physical implementation, on FPGAs, of the platform is also presented in this paper. Both the simulator and the FPGA implementation have been validated on several significant applications. This work is a major step in the definition and building of a massively parallel system on a chip.

Our next work is to implement a RTL version of both the X-Net and the mpNoC networks. This FPGA implementation will consume the empty area available in the current version. The area is one of the most important constraint encountered during the FPGA implementation. Dealing with the consumed area for an implementation, it is also necessary to deal with the choice of our initial processor: the MIPS. The MIPS processor has been chosen for our first implementation because of its open source availability in both SystemC and RTL level. However, the RTL version of the MIPS processor is not optimized for a particular FPGA, contrary to processors such as the MicroBlaze [23]. Using

MicroBlaze as an initial processor to design both the ACU and PEs could considerably reduce the consumed area and enhance the implementation effectiveness. At a first glance, the works done on the MIPS may be reproduced on the instruction set and the 5-stage pipeline of the MicroBlaze.

The major drawback of pure SIMD machines is their inability to effectively use all the system during all the computation. Typically a parallel `if/then/else` instruction deactivates half of the PEs during the execution of the `then` part and deactivates the other half during the execution of the `else`. As it was planned with the MasPar MP-3, we can imagine several ACU controlling the PEs and allowing to considerably enhance the SIMD machine effectiveness.

Finally, on the integration side, if large configuration may not be integrated on a single chip, we are considering multi-chip implementations. Connecting together on a board, those chips will be able to act like a unique SIMD machine executing a single program. The definition of a chip interface and especially the splitting of the networks on the different chips has to be studied with a special attention on the scalability of the architecture.

Acknowledgments

The authors would like to thank the students who have helped in the first implementation of the mppSoC architecture and specially Frédéric Bastien who has worked on the FPGA implementation. They would also like to thank Smail Niar who has made useful comments on drafts of this paper.

9. REFERENCES

- [1] James D. Allen and David E. Schimmel. Issues in the design of high performance SIMD architectures. *IEEE Transactions on Parallel and Distributed Systems*, 7(8):818–829, August 1996.
- [2] ARM Corporation. NEON technology. <http://www.arm.com/products/CPUs/NEON.html>, 2006.
- [3] Tom Blank. The MasPar MP-1 architecture. In *Proceedings of the IEEE Comcon Spring 1990*, pages 20–24, San Francisco, CA, February 1990. IEEE Society Press.
- [4] Pete Claydon. picoChip: A massively parallel array processor. In *Embedded Processor Forum*, San Jose, CA, May 2003.
- [5] Robert L. Davis. The Illiac IV processing element. *IEEE Transactions on Computers*, 18(9):800–816, September 1969.
- [6] H. Du, M. Sanchez-Elez, N. Tabrizi, N. Bagherzadeh, M. L. Anido, and M. Fernandez. Interactive ray tracing on reconfigurable SIMD MorphoSys. In *Design, Automation and Test in Europe Conference (DATE'2003)*, Munich, Germany, March 2003.
- [7] Simon Duquennoy, Sébastien Le Beux, Philippe Marquet, Samy Meftali, and Jean-Luc Dekeyser. MpNoC design: Modeling and simulation. In *15th IP Based SoC Design Conference (IP-SoC 2006)*, Grenoble, France, December 2006.
- [8] W. Daniel Hillis. *The Connection Machine*. The MIT Press, Cambridge, MA, 1985. Traduction française, Masson, Paris, 1988.
- [9] R. Michael Hord. *The Illiac IV the first supercomputer*. Computer Science Press, 1982.
- [10] ITRS, International Technology Roadmap for Semiconductors. Design, 2005 edition. <http://www.itrs.net/>, 2005.
- [11] Brewster A. Kahle and W. Daniel Hillis. The Connection Machine model CM-1 architecture. *IEEE Transactions on Systems, Mans, and Cybernetics*, 19(4):707–713, July 1989.
- [12] Won Kim and Russ Tuck. MasPar MP-2 PE chip: A totally cool hot chip. In *Proc. IEEE 1993 Hot Chips Symposium*, Palo Alto, CA, March 1993.
- [13] Clyde P. Kruskal and Marc Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C-32(12):1091–1098, December 1983.
- [14] Sébastien Le Beux, Vincent Gagne, El Mostapha Aboulhamid, Philippe Marquet, and Jean-Luc Dekeyser. Hardware/software exploration for an anti-collision radar system. In *The 49th IEEE International Midwest Symposium on Circuits and Systems*, San Juan, Puerto Rico, August 2006.
- [15] Ming-Hau Lee, Hartej Singh, Guangming Lu, Nader Bagherzadeh, Fadi J. Kurdahi, Eliseu M. C. Filho, and Vladimir Castro Alves. Design and implementation of the MorphoSys reconfigurable computing processor. *The Journal of VLSI Signal Processing*, 24(2-3):147–164, March 2000.
- [16] ModelSim simulation and debug environment for design. <http://www.model.com/>.
- [17] John R. Nickolls. The design of the MasPar MP-1: A cost effective massively parallel computer. In *Proc. of the IEEE Comcon Spring 1990*, pages 25–28, San Francisco, CA, February 1990. IEEE Society Press.
- [18] OpenCores. miniMIPS overview. <http://www.opencores.org/projects.cgi/web/minimips/>.
- [19] picoChip. Pc101 picoarray. <http://www.picochip.com/technology/picoarray>.
- [20] Hartej Singh, Ming-Hau Lee, Guangming Lu, Fadi J. Kurdahi, Nader Bagherzadeh, and Eliseu M. Chaves Filho. MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Transactions on Computers*, 49(5):465–481, 2000.
- [21] The SoCLib project: An open modelling and simulation platform for system on chip design. <http://soclib.lip6.fr/>.
- [22] VSIA (Virtual Socket Interface Alliance). <http://www.vsi.org/>.
- [23] Xilinx, Inc. MicroBlaze home page. <http://www.xilinx.com/microblaze>.