

# Co-Design of Massively Parallel Embedded Processor Architectures\*

Frank Hannig, Hritam Dutta, Alexey Kupriyanov, Jürgen Teich,<sup>†</sup>  
Rainer Schaffer, Sebastian Siegel, Renate Merker,<sup>‡</sup>  
Ronan Keryell,<sup>§</sup> Bernard Pottier,<sup>¶</sup>  
Daniel Chillet, Daniel Ménard, and Olivier Sentieys<sup>||</sup>

## Abstract

*In this paper, we introduce a methodology for the systematic mapping, evaluation, and exploration of massively parallel processor architectures that are designed for special purpose applications in the world of embedded computers. The investigated class of computer architectures can be described by massively parallel networked processing elements that, using today's hardware technology, may be implemented on a single chip (SoC – System on a Chip). Existing approaches for mapping computational-intensive algorithms to parallel architectures either consider the implementation in dedicated hardware or the implementation on a given supercomputer. Many intermediate solutions between these extremes are coming up ranging from fine-grained FPGAs to coarse-grained processor arrays. For these architectures, we propose in this contribution a co-design approach in the sense of designing special purpose parallel processor architectures and efficient mapping tools simultaneously.*

## 1 Introduction

Today, the steady technological progress in integration densities and modern nanotechnology will allow implementations of hundreds of 32-bit microprocessors and more on a single die. Furthermore, the functionality of the microprocessors increases continuously, e.g., by parallel process-

ing of data with low accuracy (8 bit or 16 bit) within the microprocessor. Due to these advances, massively parallel data processing has become possible in portable and other embedded systems. These devices have to handle increasingly computational-intensive algorithms like video processing (H.264) or other digital signal processing tasks (3G), but on the other hand they are subject to strict limitations in their cost and/or power budget. This kind of applications can only be efficiently realized if design tools are able to identify the inherent parallelism of a given algorithm and if they are able to map it into correctly functional, reliable, and highly optimized systems with respect to cost, performance, and energy/power consumption. But, technical analysts foresee the dilemma of not being able to fully exploit next generation hardware complexity because of a lack of mapping tools. Hence, parallelization techniques and compilers will be of utmost importance in order to map computational-intensive algorithms efficiently to these processor arrays.

At all times, there was the exigence (demands at speed, size, cost, power, etc.) to develop dedicated massively parallel hardware in terms of ASICs (Application Specific Integrated Circuits). Let us consider the area of image processing, for instance, where a cost-benefit analysis is of crucial importance: On a given input image, sequences of millions of similar operations on adjacent picture elements (pixel) (e.g., 2- or 3-dimensional filter algorithms, edge detection, Hough transformation) have to be computed within splits of a second. The use of general purpose parallel computers like MIMD or SIMD multiprocessor machines is not reasonable because such systems are too large and expensive. Such machines are also of no use in the context of mobile environments where additional criteria such as energy consumption, weight and geometrical dimensions exclude solutions with (several) general purpose processors.

In order to avoid huge area and thus cost overheads of general purpose computers, the architecture of choice is *problem- or domain-specific*. The main problem is that the development of a new architecture requires also suitable compilers and mapping tools, respectively. *Parameterizable*

\*Supported in part by the German Science Foundation (DFG) in project under contract TE 163/13-1 and ME 1625/4-1.

<sup>†</sup>Department of Computer Science 12, Hardware-Software-Co-Design, University of Erlangen-Nuremberg, Germany. {hannig, dutta, kupriyanov, teich}@cs.fau.de

<sup>‡</sup>Institute of Circuits and Systems, Department of Electrical Engineering and Information Technology, Dresden University of Technology, Germany. {schaffer, siegel, merker}@iee1.et.tu-dresden.de

<sup>§</sup>ENST Bretagne, Département Informatique, CS 83818, 29238 Plouzané Cédex, France. rk@enstb.org

<sup>¶</sup>Architectures et Systèmes, Université de Bretagne Occidentale, Brest, France. bernard.pottier@univ-brest.fr

<sup>||</sup>IRISA, University of Rennes, France. {chillet, menard, sentieys}@irisa.fr

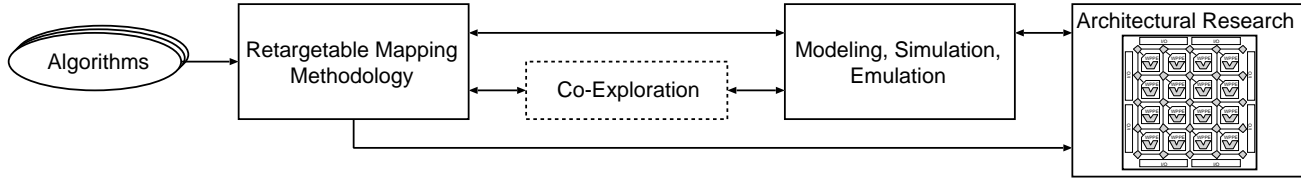


Figure 1. Co-design flow.

*architecture* and *compiler co-design* will therefore be a key step in the development of such embedded systems in the future. The main challenges are on the one hand side the extraction of common properties of regular designs independent of hardware and software, respectively. On the other side, the *analysis of program transformations and architecture parameters* is of great importance in order to achieve highly efficient and optimized systems. Concepts of *retargetable compilers* are needed for array-like architectures. One major milestone here is to study and understand the *correlation and matching of architectural parameters with the parameters of program transformations* as part of such compilers.

The interplay of the different main parts of our design methodology is depicted in Fig. 1. The parts *architectural research*, *array architecture modeling and simulation*, and *retargetable mapping methodology* are described in Section 3, Section 4, and in Section 5. But first in Section 2 we give an overview of related work.

## 2 Related Work

Properties of the considered algorithms like (piecewise) regularity, massive parallelism, and local communication between processing elements in space and time are reflected by dedicated VLSI-architectures such as proposed in [31, 33]. The processing elements of such array architectures can internally consist again of regularly connected parallel units (e.g., functional units such as adders and multipliers). Therefore, parallelism and pipelining can be combined on different hierarchical levels (e.g., task-, instruction-, and bit-level). Such architectures are also named *piecewise regular processor arrays* [31] as each processing element implements an algorithmically specialized set of functions and different functions may be computed in different regions of the array simultaneously. Here, it is assumed that the whole amount of control, especially the determination of the next instruction of each processing element, depends only on local control signals propagated through the array. Therefore, each processing element does not need to know its position within the array.

More flexible, are recently emerging coarse-grained array architectures. We summarize only some commercial architectures like the D-Fabrix [9], the DRP from NEC [22],

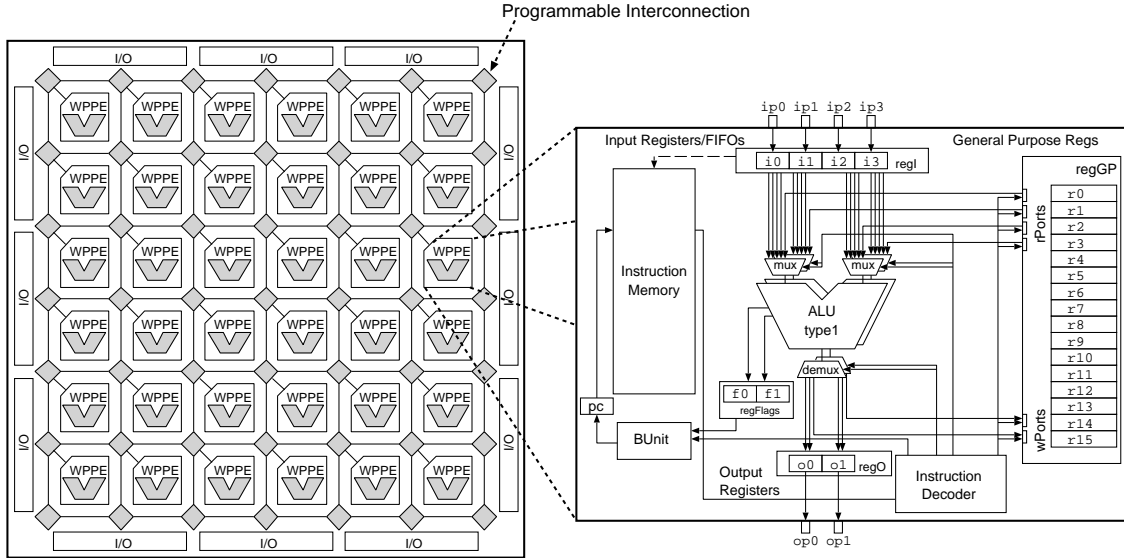
the PACT XPP [2], Bresca from Silicon Hive (Philips) [29], or QuickSilver Technology’s Adaptive Computing Machine (ACM) [24].

Both fine- and coarse-grained architectures have a lack of programmability in common, due to their own paradigms which are totally different from the *von Neumann’s*. To overcome this obstacle there exist only few synthesis tools for the design of such application specific circuits. For instance, Compaan [17] which deals with process networks or PICO-N initially developed by the Hewlett-Packard Laboratories [16, 27] and recently commercialized as PICO Express by Synfora [30]. PICO Express takes algorithm descriptions expressed in terms of sequences of nested loops and maps them to a highly optimized pipeline processor array (PPA) architecture. PICO Express supports multiple loops with streaming data and creates a single, rate matched RTL block. In the past, PICO was only able to handle perfectly nested loop programs with only static control flow. No information is available about extensions in order to handle more general loop programs. PARO [3, 23], a design system project for modeling, transformation, optimization, and processor synthesis for the class of *Piecewise Linear Algorithms* (PLA). PARO can be used during the process of automated synthesis of regular circuits.

Certainly, there exists a number of fully developed hardware design languages and tools like Handel-C [5], the SPARK environment [12], or the Nimble framework [19], but they use imperative forms as input code, some are restricted to one-dimensional array data structures and some do not even allow high-level program transformations.

## 3 Architectural Research

As target a class of massively parallel architectures is considered which we call *weakly-programmable processor arrays* (WPPA). Such architectures consist of an array of processing elements (PE) that contain sub-word processing units with only very few memory and regular interconnect structures. In order to efficiently implement a certain algorithm, each PE may implement only a certain function range. Also, the instruction set is limited and may be configured at compile-time or even dynamically at run-time. The PEs are called weakly-programmable because the control overhead of each PE is optimized and kept small. An



**Figure 2. A possible example of a WPPA with parameterizable processing elements. A WPPE consists of a processing unit which contains a set of functional units. Some functional units allow to compute sub-words in sub-word units in parallel. The processing unit is connected with input registers and an output register. A small data memory exists to temporally store computational results. One of the most important parts is the existence of an instruction sequencer as part of the control path which executes a set of control instructions from a local tiny program memory. The dashed line from the WPPE's input ports denotes a possible path for loading the instruction memory.**

example of such an architecture is depicted in Fig. 2.

The massively parallelism might be expressed by different types of parallelism: (1) several parallel working weakly-programmable processing elements (WPPEs), (2) functional and software pipelining, (3) multiple functional units within one WPPE, and finally, (4) sub-word parallelism (SWP) within the WPPEs.

### 3.1 Reconfigurable Memory Hierarchies

To adapt input and output data-flows of the mapped application to the processing elements, a WPPE is characterized by some reconfigurable capacities in the input and output registers FIFOs. This reconfiguration can be achieved through the amount and the width of the local registers and the depth of the FIFOs, in relation with the different read/write data-flows between the WPPEs. Moreover, the general purpose register files are also configurable in terms of size and width. All these configuration abilities can be exploited to significantly improve performances and save energy in the WPPEs.

Finally, we propose a segmented global memory space between the different WPPEs through a reconfigurable interconnect between each WPPE and the first level of the global memory. By extension, the reconfiguration of this interconnect can be a good candidate of energy saving by

elimination of data transfers between different memory regions (see Fig. 5).

### 3.2 WPPA Control and Utilities

At this point we consider the question of control and status of the WPPA from an outside host processor. Status involves observation (or knowledge) of program states, collectively, or individually. Control allows to put nodes into an initial state from where they can load their code and start execution.

**Local Support** As node programs does not necessarily execute the same programs, they can be reused dynamically for other code sections. Furthermore, it may be desirable to apply different behavior on different parts of the array. One way to solve these problems is to provide a fixed micro-machine that will manage the local execution, a local status, loading of instruction memory, and possible other utilities such as debugger support or exceptions. Reinitialization of a node puts the instruction pointer in a secure position from which some communication channels are scanned to find the initial sequence from a boot protocol. The micro-machine then loads the instructions into memory. Program loading can be achieved using a specific mechanism such

as a serial or parallel bus, or by using the FIFO mechanism provided for normal execution. Depending on the choice, a debugger support can be activated on receipt of control messages or by interleaving the application code with breakpoint support.

**Global Management** Loading a whole or partial program in the WPPA depends on the control mechanism.

In the case of a specific support, packets are sent to nodes by the control processor, to load the programs progressively.

In the case where no specific support is provided, the loading mechanism principle is to admit that PEs to be loaded are expecting the protocol on one of their data ports.

A network configuration data is propagated from node to node in the form of commands interpreted by the local loading mechanism (write left, write right, write memory...). This is not different from network booting mechanisms found on transputer arrays. The network program is computed on the host processor, with an obvious constraint which is the accessibility of the processor set to be loaded.

## 4 Architecture Modeling and Simulation

Since design time and cost are critical aspects during the design of processor architectures it is important to provide efficient modeling and simulation techniques in order to evaluate architecture prototypes without actually designing them. In the scope of the methodology presented here, we are looking for a flexible reconfigurable architecture in order to find out trade-offs between different architecture peculiarities for a given set of applications. Therefore, a formal description of architectures' changeable parameters is of great importance. In order to provide such a processor architecture specification we use the so-called *MACHine Markup Language* (MAML) [11] with further modifications and extensions. MAML is based on the notation of XML and used for describing the architecture parameters required by the mapping methods such as partitioning, scheduling, and functional unit and register allocation as described in Section 5.

The architectural description of an entire WPPA can be subdivided into two main abstraction levels, an *array-level* describing parameters such as the topology of the interconnection, number and location of processor and I/O-ports, etc., and a *PE-level* describing the internal structure of a WPPE. In MAML this is done by the two macro-tags *ProcElement* and *ProcArray*.

### 4.1 Array-level Specification

The array-level properties of the WPPA are described in the body of a special MAML tag called *ProcArray*. This tag specifies the parameters of the whole WPPA in general,

i.e., the name of the WPPA, the interconnect topology, the number of WPPEs, etc. For instance, if the WPPA has a grid structure of PEs the size of the WPPA must be given as the number of columns and rows. The interconnect between processor array cells is one of the very important parameters of WPPA. It is specified with the complex tag *Interconnect*, describing several peculiarities of WPPA interconnect, such as the topology or the interconnection type (bus, nearest neighbor, reconfigurability, etc.). There are many different interconnect topologies but the selection one of them is always a trade-off.

### 4.2 PE-level Specification

A schematic view of a possible WPPE model is depicted in Fig. 2. The processor element consists of three register files: input registers *regI*, output registers *regO*, and general purpose registers *regGP*. As a special register it has a *program counter* *pc* and a set of registers-flags form the register bank *regFlags*. Consequently, the registers from *regI* are given the names *i* plus the index from 0 to 3, as we have four input registers. In the same manner, the registers, which belong to *regO* can be referred with the names *o0*, *o1*. The registers of *regGP* are referred by *r0* . . . *r15*. The flag-registers are called *f0*, *f1*. Input and output registers are connected to the input and output ports: {*ip0*, *ip1*, *ip2*, *ip3*} and {*op0*, *op1*}, respectively. Reading from and writing to the registers of any register bank is established through the read- and write ports (*rPort*, *wPort*). A WPPE can have one or several functional units (FUs) of the same or different ALU types which can increase the performance of a WPPE by enabling parallel computing by the execution of VLIW instructions. FUs with a wide word width might be also configured to provide sub-word parallelism (SWP) as described in Section 4.3. An instruction decoder decodes the processor instructions stored in the instruction memory and drives the multiplexors and demultiplexors which select registers and register banks for the source and target (result) operands of FUs.

In order to distinguish between ordinary general purpose register banks and I/O register banks, the tags *InputRegisterBank* and *OutputRegisterBank* are defined. Each of them has two attributes specifying the name of the register bank and the number of registers in it. Additionally, there might not be only simple I/O registers but also FIFOs and small FSM to control the communication. The instruction memory is separated from the general memory by introduction of tag *InstructionMemory* with the memory size in bytes as attribute.

In order to provide a complete design flow, starting from the architecture specification and finishing by the compiler generation the results of compilation must be represented as binary code, so that we can put this binary code as a stimuli entry data for WPPE architecture simulation. In order

$$\begin{aligned}
h_1[x,y] &= p_i[x+2,y] - p_i[x,y] + p_i[x+2,y+2] - p_i[x,y+2] + 2(p_i[x+2,y+1] - p_i[x,y+1]) \\
h_2[x,y] &= \begin{cases} -h_1[x,y] & \text{if } C_1[x,y] \\ h_1[x,y] & \text{if } \neg C_1[x,y] \end{cases} \\
v_1[x,y] &= p_i[x,y+2] - p_i[x,y] + p_i[x+2,y+2] - p_i[x+2,y] + 2(p_i[x+1,y+2] - p_i[x+1,y]) \\
v_2[x,y] &= \begin{cases} -v_1[x,y] & \text{if } C_2[x,y] \\ v_1[x,y] & \text{if } \neg C_2[x,y] \end{cases} \\
s_1[x,y] &= h_2[x,y] + v_2[x,y] \\
s_2[x,y] &= \begin{cases} 255 & \text{if } C_3[x,y] \\ s_1[x,y] & \text{if } \neg C_3[x,y] \end{cases} \\
p_o[x,y] &= s_2[x,y] \\
C_1[x,y] &= (h_1[x,y] < 0) \\
C_2[x,y] &= (v_1[x,y] < 0) \\
C_3[x,y] &= (s_1[x,y] > 255)
\end{aligned}$$

**Figure 3. Edge detection algorithm written in the notation of a DPLA, with the iteration domain**

$$\mathcal{I} = \{x, y \in \mathbb{Z} \mid 1 \leq x \leq N-3 \wedge 1 \leq y \leq M-3\}.$$

to handle that, the MAML description is extended by an instruction image binary coding.

### 4.3 Sub-Word Parallelism in the WPPEs

Sub-word parallelism allows the parallel execution of equal instructions with low data word width (e.g. four additions with 16 bit data) on functional units with high data word width (e.g. 64 bit adder). Also the execution of complex instructions (e.g. multiply and add,  $y = \sum_i a_i \cdot b_i$ ) with multiply data input (4 data pairs with 16 bit word width) and single data output (1 date with 64 bit word width) is possible. Consequently the type of SWP (operation and number of sub-words) have to be given in the description for each FU.

Mostly the sub-words, which are packed together in full length words, have to be rearranged for the next calculation. Therefore new instructions are needed, which are called packing instructions. The packing instructions can be added to given FUs or extra FUs will be described. Because the sub-words are stored in the full length words and the rearranging will be done in the FUs, no additional parameters are needed for the registers description.

### 4.4 Simulation

In order to evaluate different processor architectures and to find out which of them optimally fulfills the given requirements, a simulation of the whole parallel processor architecture is needed. The architecture simulation can be done on different levels. On one hand, register-transfer (RT) level generally enables very flexible, precise but sometimes relatively slow simulation, on the other hand, instruction-set (IS) level performs a very fast simulation but not that precise and flexible. Prominent feature of massively parallel

processor architectures is a high number of registers. The RT level high-speed simulation approach presented in [18] is applied here as it enables high-speed RTL simulation of complex architectures with a large amount of registers, such as existent in processor arrays. This simulation methodology provides a direct automatic generation of the simulator from the given RTL netlist.

## 5 Retargetable Mapping Methodology

Starting point of our mapping methodology is a given nested loop program in *single assignment code* (SAC), where the whole parallelism is explicitly expressed. SAC is closely related to a set of recurrence equations, a formalism introduced by Karp, Miller, and Winograd [15]. This formalism has been used in many languages and advanced over the years about affine dependencies and piecewise definitions. E.g., *Systems of Affine Recurrence Equations* (SARE) which are used in the Alpha language [34, 35], the class of *Affine Indexed Algorithms* (AIA) [8], the class of *Piecewise Linear Algorithms* (PLA) [31], and recently, an algorithm class with run-time dependent conditionals, *Dynamic Piecewise Linear Algorithms* (DPLA) [13]. An example for an edge detection algorithm written as DPLA is given in Fig. 3.

### 5.1 Parameter Matching

In order to map a given loop program to a specific architecture, the architectural parameters have to be considered during the loop transformations, i.e., the architectural parameters have to be matched with the program transformation parameters of the mapping method. In Fig. 4, a brief overview of the relation between loop transformations and architectural parameters is given. For instance, in or-

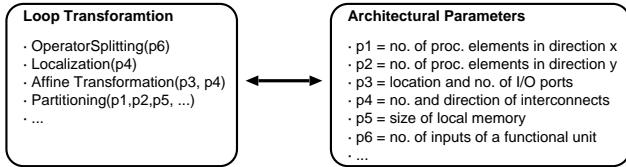


Figure 4. Parameter matching.

der to adapt a given algorithm to an architecture, complex statements have to be split into operations with only  $p6$  operands according to the number of possible inputs of a functional unit. Affine transformations might be used to fit algorithm dependencies to the array topology or to transform input/output variables to the physical location of I/Os. Partitioning of loop programs is done according to constraints imposed by the physical hardware. Here, the tiling matrices have to be chosen according to the size of the array, the local memory, I/O-bandwidths etc. How this can be done is briefly described in the next section.

## 5.2 Hierarchical Partitioning

The processor array architectures are characterized by a given constant amount of local memory for each WPPE and a hierarchical cache architecture. The purpose of parameter matching is to apply transformations in the polytope model in order to obtain a processor array implementation with optimal use of local memory and the given cache structure. Simple partitioning methods are known as LPGS- and LSGP-partitioning. They allow only the matching of either the number of processing elements or the amount of local memory respectively. Co-partitioning, a hierarchical partitioning method, combines both schemes and thereby allows to balance memory, communication, and the size of the array [8]. One way to either optimize or match a memory hierarchy is described in [28], where we find optimal tiling matrices in two steps. An outer partitioning (see Fig. 5) minimizes the communication between the array and the peripheral memory through optimal data-reuse within a memory hierarchy. An inner partitioning matches a fixed number of processing elements. The summary of another approach is first to determine the optimal tile shape given the set of dependencies. Then a selection of a set of optimal outer tile sizes from available information of memory sizes and communication rate is performed. Subsequently, a determination of a set of inner tiles is done by dividing the outer tile size by the given processor array dimensions. Finally, the local memory usage for all inner tiles is calculated. The inner tile with most optimal usage of local memory is selected and henceforth the outer tile [7]. In addition, the problem of automated controller generation for a hierarchically partitioned algorithms has been tackled successfully in [7].

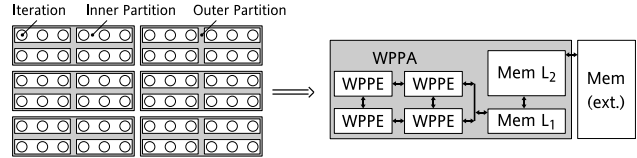


Figure 5. Co-partitioning of an iteration space and the corresponding WPPA with a memory hierarchy.

## 5.3 Scheduling and Allocation

In consideration of the architectural parameters, the allocation on array level is done by partitioning the iteration space as described in Section 5.2. Since, the control overhead should be kept small we investigate static scheduling techniques which simultaneously optimize the schedule between WPPEs (global schedule) and within WPPEs (local schedule). In order to handle run-time dependent conditionals we extended our algorithm class (s. Section 5) and studied speculative scheduling techniques based on mixed integer linear programming to achieve highest performance and utilization [14]. Considering also SWP, simultaneous scheduling can become a crucial task. Therefore, we investigate also hierarchical scheduling methods, where we combine ILP techniques for the global schedule with the following mapping methodology on WPPE level.

A key feature of weakly-programmable array architectures is the parallel processing of sub-words (SWs) in the functional units of the WPPEs. The sub-word units<sup>1</sup> can be considered as processing elements of an array-like architecture. This view allows a new approach for exploiting SWP [25, 26]. Design methods used for array architecture design will be adapted to the constraints of SWP with the following objectives: (1) A high degree of parallel processing shall be obtained by mapping the individual operations of the algorithm to the sub-word units of the processing unit. Thus, the speed-up will be achieved primarily in relation to conventional implementations. (2) The process of packing the sub-words to words and unpacking the words after the computation must be performed, with the aim of minimization of the execution time and the power consumption (number of memory accesses). Lower power consumption compared with conventional implementations and an optimal utilization of the available registers shall be ensured by an optimal data-reuse.

The mapping methodology will be based on the new general algorithm transformation methodology (embedding  $\rightarrow$  partitioning  $\rightarrow$  localization) [28, 32]. We will refine the partitioning step for the extensive usage of the SWP properties.

<sup>1</sup>During the processing of sub-words in a FU, the unit is split in several sub-word units, which are working in parallel.

With the method of co-partitioning the operations of the algorithm can be mapped to SW units and scheduled. The restrictions of the target architecture, such as the number of SW units, the number of available registers, the parameters of the memory hierarchy and the SWP instructions set, must be taken into account during the partitioning step. There-with, the reuse of variables and the packing processes will be arranged substantially.

## 5.4 Memory Optimization

Memory usage in a massively parallel SoC must be considered with the most importance since each WPPA does not have a lot of memory per WPPE. Furthermore, the computation involved in address computation needs to be decreased as much as possible. Unfortunately, these two objectives can be contradictory since minimizing memory usage can involve complex address computations to compress the memory space.

In order to analyze the memory usage at different levels (interprocedural or instruction levels) in the algorithms, we use the PIPS source-to-source transformer and analyzer framework. Since using the global data-flow graph at the program level is too complex for real interprocedural programs, we use region analysis to abstract the data dependences on polyhedral sets of array elements [6] combined with a precise semantical analysis on the integer variables. This array regions are used to compute the communication operators and the temporary variables. Once array privatization has been applied, some more aggressive methods can be used to increase the memory reuse by using also a linear algebra formalism [4].

At the architecture level, we plan to model the different levels of memories and parallelism (array of WPPE with some memory banks and inside each WPPE the sub-word parallelism) by using the linear algebra based representation presented in [1] for HPF (High Performance Fortran) compilation: the physical WPPE geometry is directly mapped onto the HPF model and the sub-word parallelism is presented as a new dimension. Memory and communication optimizations can be solved at higher level at once before using more local optimizations.

## 5.5 Bit-width Optimization

Processors with Sub-Word Parallelism (SWP) capabilities, can process data with different bit-widths. Different data-types can be manipulated by the functional units. With these SWP capabilities, the data level parallelism can be exploited to optimize the implementation. On the one hand, the data memory size and the code execution time can be reduced by diminishing the data bit-width. Indeed, an operator is split to execute several operations in parallel on a same operator with reduced data bit-width. This technique

can accelerate the code execution time up to a factor  $k$  corresponding to the number of operations executed in parallel on a same operator. In [10], this technique has been used to implement a CDMA synchronization loop in the TigerSharc DSP. The SWP capabilities enable to achieve 6.6 MAC per cycle with two MAC units. On the other hand, the computation accuracy can be increased by using data with higher bit-width. In this case, only one operation is executed per operator. To optimize the implementation, a trade-off between the memory size, the code execution time and the computation accuracy must be achieved.

To respect the application quality criteria, a minimum computation accuracy must be fulfilled. Thus, a methodology which optimizes the implementation for a given accuracy constraint has been proposed in [20]. The SWP instructions which minimize the code execution and the memory size, and fulfill the accuracy constraint are selected. In [21], this approach enables to accelerate the execution time of a WCDMA rake receiver on a TI C64x DSP up to a factor of 3.5, compared to the classical implementation without SWP instructions. The methodology optimizes the fixed-point implementation while respecting the quality criteria of the WCDMA application.

## 6 Conclusions and Outlook

We have introduced a co-design methodology for the systematic modeling, simulation, mapping, and evaluation of massively parallel processor architectures that are designed for special purpose applications in the world of embedded computers. The next project's steps will include the development of architecture and compiler prototypes. In the future, we would like to study the transformation order to derive optimal co-designs and we want to develop further transformations to match the architectural parameters with the parameters of our retargetable mapping methodology. The long-term objective of the project is the automatic co-exploration of architectures and suited mapping transformations. Here, it has to be investigated which transformations must be applied in order to efficiently use the features of a given architecture. Here, beside the described efficient simulation also emulation is a must for enabling design space exploration.

## References

- [1] C. Ancourt, F. Coelho, F. Irigoien, and R. Keryell. A Linear Algebra Framework for Static HPF Code Distribution. *Scientific Programming*, 6(1):3–27, Spring 1997. Special Issue — High Performance Fortran Comes of Age.
- [2] V. Baumgarte, G. Ehlers, F. May, A. Nüchel, M. Vorbach, and M. Weinhardt. PACT XPP – A Self-Reconfigurable Data Processing Architecture. *The Journal of Supercomputing*, 26(2):167–184, 2003.

- [3] M. Bednara and J. Teich. Automatic Synthesis of FPGA Processor Arrays from Loop Algorithms. *The Journal of Supercomputing*, 26(2):149–165, 2003.
- [4] Y. Bouchebaba and F. Coelho. Tiling and Memory Reuse for Sequences of Nested Loops. In *8th International Euro-Par Conference, Euro-Par 2002*, number 2400 in Lecture Notes in Computer Science, pages 255–264, Paderborn, Germany, Aug. 2002.
- [5] CELOXICA, Handel-C. [www.celoxica.com](http://www.celoxica.com).
- [6] B. Creusillet and F. Irigoien. Interprocedural analyses of Fortran Programs. *Parallel Computing*, 24(3–4):629–648, 1998.
- [7] H. Dutta. Mapping of Hierarchically Partitioned Regular Algorithms onto Processor Arrays. Master’s thesis, University of Erlangen-Nuremberg, 2004.
- [8] U. Eckhardt and R. Merker. Hierarchical Algorithm Partitioning at System Level for an Improved Utilization of Memory Structures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(1):14–24, 1999.
- [9] Elixent Ltd. [www.elixent.com](http://www.elixent.com).
- [10] D. Esfathiou, J. Fridman, and Z. Zvonar. Recent Developments in Enabling Technologies for the Software-Defined Radio. *IEEE Communication Magazine*, 37(8):112–117, August 1999.
- [11] D. Fischer, J. Teich, M. Thies, and R. Weper. Design Space Characterization for Architecture/Compiler Co-Exploration. In *ACM SIG Proceedings International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES 2001)*, pages 108–115, Atlanta, GA, U.S.A., November 2001.
- [12] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. SPARK: A High-Level Synthesis Framework for Applying Parallelizing Compiler Transformations. In *Proceedings of the International Conference on VLSI Design*, Jan. 2003.
- [13] F. Hannig and J. Teich. Dynamic Piecewise Linear/Regular Algorithms. In *Proceedings of the Fourth International Conference on Parallel Computing in Electrical Engineering (PARELEC 2004)*, pages 79–84, Dresden, Germany, Sept. 2004.
- [14] F. Hannig and J. Teich. Resource Constrained and Speculative Scheduling of an Algorithm Class with Run-Time Dependent Conditionals. In *Proceedings of the 15th IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP 2004)*, pages 17–27, Galveston, TX, U.S.A., Sept. 2004.
- [15] R. Karp, R. Miller, and S. Winograd. The Organization of Computations for Uniform Recurrence Equations. *J. of the Association for Computing Machinery*, 14(3):563–590, 1967.
- [16] V. Kathail, S. Aditya, R. Schreiber, B. R. Rau, D. C. Cronquist, and M. Sivaraman. Pico: Automatically Designing Custom Computers. *Computer*, 35(9):39–47, 2002.
- [17] B. Kienhuis, E. Rijpkema, and E. Deprettere. Compaan: Deriving Process Networks from Matlab for Embedded Signal Processing architectures. In *Proc. CODES’00, the 8th Int. Workshop on Hardware/Software Co-Design*, San Diego, U.S.A., May 2000.
- [18] A. Kupriyanov, F. Hannig, and J. Teich. High-Speed Event-Driven RTL Compiled Simulation. In *Proceedings of the 4th International Samos Workshop on Systems, Architectures, Modeling, and Simulation (SAMOS 2004)*, Island of Samos, Greece, July 2004.
- [19] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood. Hardware-software Co-Design of Embedded Reconfigurable Architectures. In *37th Design Automation Conference*, pages 507–512, Los Angeles, CA, June 2000.
- [20] D. Menard, D. Chillet, F. Charot, and O. Sentieys. Automatic Floating-point to Fixed-point Conversion for DSP Code Generation. In *ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES 2002)*, pages 270–276, Grenoble, France, October 2002.
- [21] D. Menard, M. Guitton, P. Quemerais, and O. Sentieys. Efficient Implementation of a WCDMA Rake Receiver on the TMS320C64x. In *37th Asilomar Conference on Signals, Systems and Computers*, Monterey, US, Nov. 2003.
- [22] M. Motomura. A Dynamically Reconfigurable Processor Architecture. In *Microprocessor Forum*, CA, 2002.
- [23] PARO Design System Project. [www12.informatik.uni-erlangen.de/research/paro](http://www12.informatik.uni-erlangen.de/research/paro).
- [24] QuickSilver Technology. [www.qstech.com](http://www.qstech.com).
- [25] R. Schaffer, R. Merker, and F. Catthoor. Systematic Design of Programs with Sub-Word Parallelism. In *International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002)*, pages 393–298, Sept. 2002.
- [26] R. Schaffer, R. Merker, and F. Catthoor. Causality Constraints for Processor Architectures with Sub-Word Parallelism. In *EUROMICRO Symposium on Digital System Design (DSD 2003)*, pages 82–89, Sept. 2003.
- [27] R. Schreiber, S. Aditya, B. Rau, V. Kathail, S. Mahlke, S. Abraham, and G. Snider. High-Level Synthesis of Non-programmable Hardware Accelerators. Technical Report HPL-2000-31, Hewlett-Packard Laboratories, Palo Alto, May 2000.
- [28] S. Siegel and R. Merker. Optimized Data-Reuse in Processor Arrays. In *Proceedings of the 15th IEEE International Conference on Application-specific Systems, Architectures, and Processors (ASAP 2004)*, pages 315–325, Sept. 2004.
- [29] Silicon Hive. [www.siliconhive.com](http://www.siliconhive.com).
- [30] Synfora, Inc. [www.synfora.com](http://www.synfora.com).
- [31] J. Teich. *A Compiler for Application-Specific Processor Arrays*. PhD thesis, Institut für Mikroelektronik, Universität des Saarlandes, Saarbrücken, Deutschland, 1993.
- [32] J. Teich and L. Thiele. Exact Partitioning of Affine Dependence Algorithms. In E. Deprettere, J. Teich, and S. Vassiliadis, editors, *Embedded Processor Design Challenges*, volume 2268 of *Lecture Notes in Computer Science (LNCS)*, pages 135–153, Mar. 2002.
- [33] J. Teich, L. Thiele, and L. Zhang. Scheduling of Partitioned Regular Algorithms on Processor Arrays with Constrained Resources. *J. of VLSI Signal Processing*, 17(1):5–20, Sept. 1997.
- [34] H. L. Verge, C. Mauras, and P. Quinton. The ALPHA language and its use for the design of systolic arrays. *J. VLSI Signal Process. Syst.*, 3(3):173–182, 1991.
- [35] D. Wilde and O. Sié. Regular Array Synthesis using Alpha. In *Int. Conf. on Application Specific Array Processors, San Francisco, California*, pages 200–211, Aug. 1994.