Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 794753, 20 pages
doi:10.1155/2012/794753

Research Article

Communication and Memory Architecture Design of Application-Specific High-End Multiprocessors

Yahya Jan and Lech Jóźwiak

Faculty of Electrical Engineering, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Correspondence should be addressed to Yahya Jan, y.jan@tue.nl

Received 12 August 2011; Revised 27 November 2011; Accepted 5 January 2012

Academic Editor: Menno M. Lindwer

This paper is devoted to the design of communication and memory architectures of massively parallel hardware multiprocessors necessary for the implementation of highly demanding applications. We demonstrated that for the massively parallel hardware multiprocessors the traditionally used flat communication architectures and multi-port memories do not scale well, and the memory and communication network influence on both the throughput and circuit area dominates the processors influence. To resolve the problems and ensure scalability, we proposed to design highly optimized application-specific hierarchical and/or partitioned communication and memory architectures through exploring and exploiting the regularity and hierarchy of the actual data flows of a given application. Furthermore, we proposed some data distribution and related data mapping schemes in the shared (global) partitioned memories with the aim to eliminate the memory access conflicts, as well as, to ensure that our communication design strategies will be applicable. We incorporated these architecture synthesis strategies into our quality-driven model-based multi-processor design method and related automated architecture exploration framework. Using this framework, we performed a large series of experiments that demonstrate many various important features of the synthesized memory and communication architectures. They also demonstrate that our method and related framework are able to efficiently synthesize well scalable memory and communication architectures even for the high-end multiprocessors. The gains as high as *12-times* in performance and *25-times* in area can be obtained when using the hierarchical communication networks instead of the flat networks. However, for the high parallelism levels only the partitioned approach ensures the scalability in performance.

1. Introduction

The recent spectacular technology has enabled implementation of very complex multi-processor systems on single chips (MPSoCs). Due to this rapid progress, the computational demands of many applications, which required hardware solutions in the past, today can be satisfied by software executed on micro-, signal-, graphic-, and other processors. However in parallel, new highly demanding embedded applications are emerging, in fields like communication and networking, multimedia, medical instrumentation, monitoring and control, military, and so forth, which impose stringent and continuously increasing functional and parametric demands. The demands of these applications cannot be satisfied by systems implemented with general-purpose processors (GPPs). For these highly demanding applications, increasingly complex and highly optimized application-specific MPSoCs are required. They have to perform real-time computations to extremely tight schedules, while satisfying high demands regarding the energy, area, cost, and development efficiency. High-quality MPSoCs for these applications can only be constructed through usage of efficient application-specific system architectures exploiting more adequate concepts of computation, storage, and communication, as well as usage of efficient design methods and electronic design automation (EDA) tools [1].

Some of the representative examples of these highly demanding applications include the based-band processing in wired/wireless communication (e.g., the upcoming 4G wireless systems), different kinds of encoding/decoding in communication, image processing and multimedia, 3D graphics,

ultrahigh definition television (UHDTV), encryption applications, and so forth. These applications require to perform complex computations with a very high throughput, while at the same time demanding low energy and low cost. The decoders of the low density parity check (LDPC) codes [2], adopted as an advance error-correcting scheme in the newest wired/wireless communication standards, like IEEE 802.11n, 802.16e/m, 802.15.3c, 802.3an, and so forth, for applications as digital TV broadcasting, mm-wave WPAN, and so forth, can serve as a representative example of such applications. These standards specify ultrahigh throughput figures in the range of Gbps and above [3] that cannot be achieved using general-purpose processors (GPPs), digital signal processors (DSPs) [4], or general-purpose graphic processing units (GPGPUs) [5]. For example, an execution of LDPC decoding on the famous Texsas Instruments TMS320C64xx DSP processor running at 600 MHz delivers a throughput of only 5 Mbps [4]. Similarly, implementations of LDPC decoders on the multicore architectures result in throughputs in the order of 1~2 Mbps on the general-purpose x86 multicores, and ranging from 40 Mbps on the GPU to nearly 70 Mbps on the CELL broadband engine (CELL/B.E) as reported in [5]. For the realization of the throughput as high as several Gbps, massively parallel hardware multiprocessors are indispensable.

Traditional hardware accelerator design approaches are focused on an effective design of data processing modules, without adequately taking into account the memory and communication structure design [6]. However, for the applications that require massively parallel hardware implementations, the effectiveness of communication and memory architectures and the compatibility of the processing, memory, and communication subsystems play the decisive role. As we will demonstrate in this paper, the communication architectures cannot be designed as simple flat homogenous networks and the memory as a simple (multi-port) memory. The communication network among the processors or processors and memories has a dominating influence on all the most important physical design aspects, such as delay, area, and power consumption. The additional performance gains expected from an increased parallelism will end up in diminished returns, when exploding the interconnect complexity. Therefore, all the architectural as well as the data and computation mapping decisions regarding the memories and processors have to be made in the context of the communication architecture design to actually boost the performance. For the massively parallel hardware accelerators, the problem of how to keep up with the increasing processing parallelism while ensuring the scalability of memory and communication is a very challenging design problem. To our knowledge, it has not been addressed satisfactorily till now.

This paper is devoted to the design of communication and memory architectures of application-specific massively parallel hardware multiprocessors. First, it discusses the communication and memory-related design issues of such multiprocessors. Analysis of these issues resulted in adequate architecture concepts and design strategies for their solutions. These concepts and strategies have been incorporated to our quality-driven model-based accelerator design methodology [6] and related automatic architecture design space exploration (DSE) framework. This makes it possible to effectively and efficiently resolve the memory and communication design problems, and particularly, to ensure the scalability of the corresponding architectures. We exploit these strategies in a coherent manner when at the same time accounting for the corresponding task and data mapping to particular processors and memories, as well as the technology-related interconnect and memory features, such as delay, power dissipation or area, and tradeoffs among them.

As a representative test case, we use LDPC decoders for the above-mentioned newest communication system standards. We demonstrate the application of these design strategies to the design of the multi-processor LDPC decoders. Using our DSE framework, we performed a large series of experiments with the design of various multi-processor accelerators, when focusing on their communication and memory architectures. In this paper, we discuss a part of our results from these experiments.

The rest of the paper is organized as follows. Section 2 discusses the memory and communication-related issues of hardware multiprocessors. Section 3 introduces our quality-driven model-based multi-processor design methodology. Section 4 discusses our approaches to design the efficient communication and memory architectures and related experimental results. Section 5 presents the main conclusions of the paper.

2. Issues and Requirements of Communication and Memory Architecture Design for High-End Multiprocessors

Hardware acceleration of critical computations has been intensively researched during the last decade, mainly for signal, video, and image processing applications, for efficiently implementing transforms, filters, and similar complex operations. This research was focused on the monolithic processing unit synthesis with the so-called "high-level synthesis" (HLS) methods [7–14], and not on the massively parallel multi-processor accelerators required for the high-end applications. Specifically, this research did not address the memory and communication architecture design of multi-processor accelerators. HLS only accounts for a simple memory in the form of registers and simple flat interconnect structure between the data path functional units and registers.

Although some research results related to the memory and communication architectures can be found in the literature [15–21] in the context of programmable on-chip multiprocessor systems, the memory and communication architectures were proposed there for the much larger and much slower programmable processors. They are not adequate for the small and ultra-fast hardware processors of the massively parallel multi-processor accelerators, due to a much too low bandwidth and scalability issues. The approaches proposed in the context of the programmable on-chip multiprocessors utilize time-shared communication resources, such as shared buses or network on chip (NoC). Such communication

resources are however not adequate to deliver the data transfer bandwidth required for the massively parallel multi-processor accelerators. In case of the massively parallel multiprocessor accelerators, the application-specific processors and the corresponding memory and communication architectures must be compatible (match each other) in respect to bandwidth (parallelism). Therefore, the communication architecture cannot be realized using the traditional NoC or bus communication to connect the processing and storage resources but requires point-to-point (P2P) communication architectures compatible with the parallel processing and memory resources. The traditional NoC-based communication architectures utilize a network of switches, as for instance, each switch connected to one resource (processor, memory, etc.) and four interconnected neighboring switches forming a mesh [20]. This way a large number of resources can be connected without using long global wires and thus reducing the wire delays (scalability). However, the time-shared links introduce extracommunication cycles, which negatively impact the communication and overall performance. The performance degradation grows with the increase of the number of processing elements and more global or irregular application communication patterns and grows especially fast for applications that require a large number of processors and massive global or irregular communication. Our approach to communication architecture is somewhat similar to the approaches proposed in [15, 16], but only in relation to the concept of hierarchical organization of the computation and communication resources, while this concept is differently exploited in our case. Moreover, these approaches consider memory sharing limited to a cluster of processors, but do not consider the global memories shared among the processing tiles.

Since LDPC decoding is used as a representative application in the evaluation of our design method, as well as our memory, and communication architectures, we briefly discuss the processor, memory and communication architectures proposed for the LDPC decoding. In the past, several partially parallel architectures have been proposed in the past for the LDPC decoding [22–30]. However, they only deliver a throughput of a few hundreds of Mbps. For the so low throughput, a very limited processing parallelism is exploited, and in consequence, simple communication and memory architectures are needed in the form of simple shifters and vector memories, correspondingly. The proposed partially parallel architectures are not adequate for the high-end applications that require throughputs in the ranges of multi-Gbps. To achieve such ultrahigh throughput, massive parallelism has to be exploited. This makes the memory and communication architecture design a very challenging task.

From the above discussion of the related research, it follows that the memory and communication architecture design, being of crucial importance for the high-end hardware multiprocessors, is not adequately addressed by the related research.

Many modern applications (e.g., various communication, multimedia, networking, or encryption applications, etc.) involve sets of heterogeneous data-parallel tasks with complex intertask data dependencies and interrelationships between the data and computing operations at the task level. Often the tasks iteratively operate on each other data. One task consumes and produces data in one particular order, while another consumes and produces data in a different order. Additionally, in the high-performance multiprocessor accelerators, parallelism has to be exploited on a massive scale. However, due to area, energy consumption, and cost minimization requirements, partially parallel architectures are often used which are more difficult to design than the fully parallel ones. Moreover, many of the modern applications involve algorithms with massive data parallelism at the macro-level or task-level functional parallelism. To adequately serve these applications, hardware accelerators with parallel multi-processor macroarchitectures have to be considered. These macroarchitectures have to involve several identical or different concurrently working hardware processors, each operating on a (partly) different data subset. This all results in complex memory accesses and complex communication between the memories and processing elements. For applications of this kind, the main design problems are related to an adequate resolution of memory and communication bottlenecks and to decreasing the memory and communication hardware complexity.

Moreover, each of the processors of the multi-processor can be more or less parallel. This results in the necessity to explore the various possible tradeoffs between the parallelism at the micro- and macroarchitecture levels. The two architecture levels are strongly interwoven also through their relationships with the memory and communication structures. Each micro-/macroarchitecture combination affects the memory and communication architectures in a different way. For example, exploitation of more data parallelism in a computing unit microarchitecture usually demands getting the data in parallel for processing. This requires simultaneous access to memories in which the data reside (this results in for example, vector, multibank, or multi-port memories) and simultaneous transmission of the data (this results e.g., in multiple interconnects), or prefetching the data in parallel to other computations. This substantially increases the memory and communication hardware. From the above, it should be clear that for applications of this kind, complex interrelationships exist between the computing unit design and corresponding memory and communication structure design. Also, complex tradeoffs have to be resolved between the accelerator effectiveness (e.g., computation speed or throughput) and efficiency (e.g., hardware complexity and power consumption).

The traditionally used simple flat communication scheme, independent of its specific implementation, does not scale well with the increase in the number of processing elements and/or memories. For instance, in the switch-based architectures, both the switch complexity and the number of switches grow with the increase of the number of processing elements and/or memories. In the traditional flat interconnection scheme, for n processing elements that have to communicate with m memories, we require an $m \times n$ (*Inputs Ports* $\times$ *Outputs Ports*) crossbar switch, as shown in Figure 1. In result, when used for a massively parallel hardware

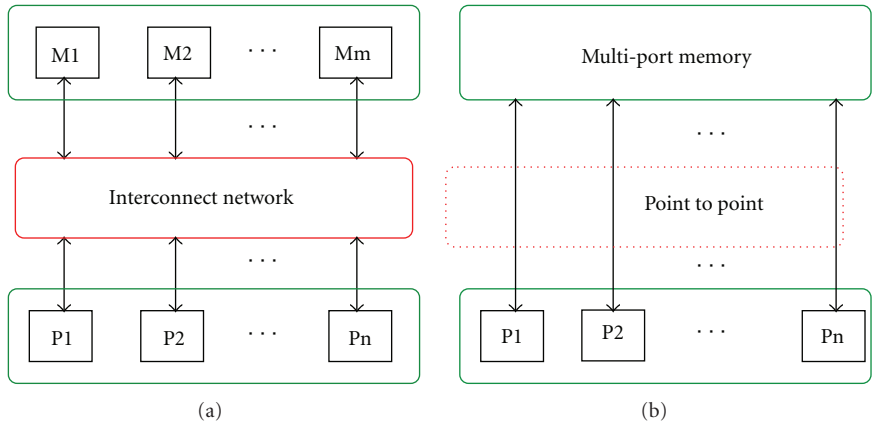

Figure 1: (a) Example of communication network among M global memories and N processors (b) Multi-port memory structure to satisfy the bandwidth requirement of multiprocessors with low complexity point-to-point (P2P) interconnects.

multiprocessor, the communication network influence usually dominates the processing elements influence on the throughput, circuit area, and energy consumption. Finally, the large flat switch that would be necessary for such a multiprocessor accelerator can be difficult to place and route, even with the most advanced synthesis tools. The place and route may use a long time or in some cases not finish their work at all. This represents an actual practical limitation on the interconnect design.

Regarding the memory issues, the memory bandwidth (number of ports) should be compatible with the processing bandwidth. Thus, a multi-port memory application with as many memory ports as required by the processing elements (aggregate bandwidth) seems to be the most natural and straightforward approach (see Figure 1). However, with increase in the processing parallelism, the required memory bandwidth (number of ports) increases. The situation quickly deteriorates with parallelism increase resulting in a high complexity due to high memory bandwidth (number of ports) required in the massive parallelism range. For the massively parallel multiprocessors, the single multi-port memory would have a prohibitively large area and long delay, when satisfying the required memory bandwidth (see Figure 2). Therefore, the data have to be organized in multiple multibank or vector memories to satisfy the required memory bandwidth, while keeping the delay and area of the memory architecture substantially lower. Consequently, the most important issues of the memory architecture design are the following:

(i) the organization of data in vectors (tiles) and the data tiles into multiple memory tiles (partitions) to satisfy the required bandwidth,

(ii) the data distribution and related data mapping into the memory tiles ensuring the conflict-free memory accesses and reducing the memory-processor communication complexity.

It is possible that a data distribution scheme would be conflict-free, but data might be distributed very randomly in the memory partitions. This would increase the communication complexity. Therefore, a memory exploration and synthesis method should adequately address the issues of memory partitioning and data distribution. Also, with increase of the processing parallelism, data have to be partitioned and stored in more and more distributed parallel memories for more parallel access. This causes the memory block sizes to shrink. At some point, it becomes not any more efficient to store the data in embedded SRAM memories, but the register-based (Flip-Flop) memories have to be used which are more efficient for small memory sizes. We take into account this issue during the memory architecture design. Our experiments with different memory configurations demonstrated that for sizes lower than (height $\times$ width $=$ 32×168), the SRAM memories are less efficient than the FF-based memories. For example, a memory of size (16×168)

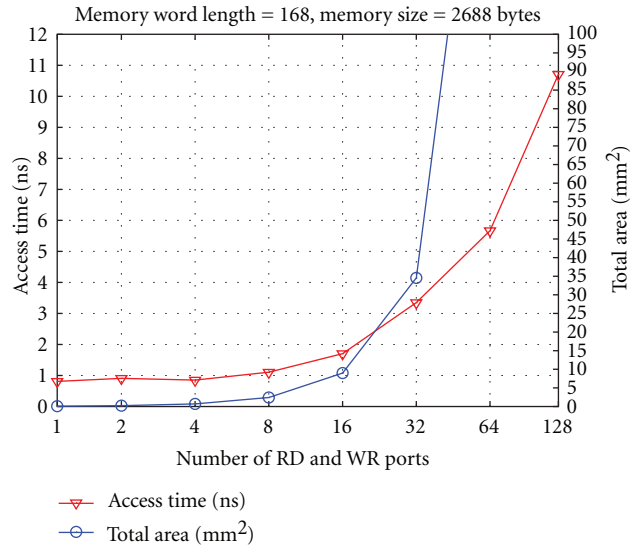

Figure 2: Area versus access time of multi-port memory characterized for CMOS 90 nm process using HP CACTI 5.3 cache/memory compiler.

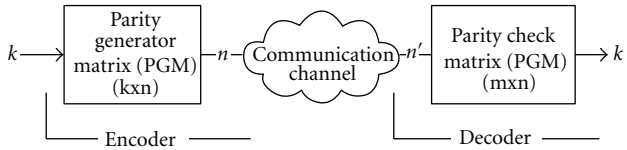

Figure 3: LDPC encoding and decoding process.

when implemented as embedded SRAM is almost *1.6 times* larger than when implemented as FF-based (implemented in TSMC 90 nm LPHP Standard Cell Library) memory, and the area proportion grows fast with further decrease in memory sizes. Therefore, for the case of IEEE 802.15.3c LDPC decoders, the SRAM-based memories are only efficient (and considered in our DSE and experimental designs) for a combined processing parallelism of up to 84 only.

Additionally, the memory and communication issues are not orthogonal in nature, resolving and optimizing one issue in separation heavily influences the other. Thus, the memory and communication architecture synthesis has to be realized as one coherent synthesis process accounting for the mutual influences and tradeoffs.

Summing up, the massive data, operation-level and task-level parallelism to be exploited to achieve the ultra-high throughput required by the highly demanding applications, the complex interrelationships between the data and computing operations, and the combined parallelism exploitation at the two architecture levels (micro-/macro-architecture) make the design of an effective and efficient communication and memory architecture a very challenging task. To effectively perform this task, the (heterogeneous) parallelism available in a given application has to be explored and exploited in an adequate manner in order to satisfactorily fulfill the design requirements through constructing an architecture that satisfies the required performance, area, and power tradeoffs.

To illustrate the requirements and issues of memory and communication architecture design, as well as to introduce and illustrate our design approach, we will use a representative case of the low-density parity-check code (LDPC) decoding.

A systematic LDPC encoder encodes a message of k bits into a codeword of length n with the message bits k followed by m parity checks, as shown in Figure 3. Each parity check is computed based on a subset of message bits. The codeword is transmitted through a communication channel to a decoder. The decoder checks the validity of the received codeword by computing these parity checks using a parity check matrix (PCM) of size $m \times n$. To be valid, a codeword must satisfy the set of all m parity checks. In Figure 4, an example PCM for a (7,4) LDPC code is given. "1" in a position $\text{PCM}_{i,j}$ of this matrix means that a particular bit participates in a parity check equation. Each PCM can be represented by its corresponding bipartite graph (Tanner graph). The Tanner graph corresponding to an (n, k) LDPC code consists of n variable nodes (VNs) and $m = n - k$ check nodes (CNs), connected with each other through edges, as shown in Figure 4. Each row in the parity check matrix represents a parity check equation c_i, $0 \le i \le m - 1$, and each column represents a coded bit v_j, $0 \le j \le n - 1$. An edge exists between a CN i and VN j if the corresponding value $\text{PCM}_{i,j}$ is nonzero in the PCM.

Usually, iterative message passing (MP) algorithms are used for decoding of the LDPC codes [31]. The algorithm starts with the so-called intrinsic log-likelihood ratios (LLRs) of the received symbols based on the channel observations. During decoding, specific messages (extrinsic) are exchanged among the check nodes and variable nodes along the edges of the corresponding Tanner graph for a number of iterations. The variable and check node processors (VNP, CNP) corresponding to the VN and CN computations iteratively update each other data, until all the parity checks are satisfied or the maximum number of iterations is reached. The data related to the check and variable node computations are stored in the corresponding shared check and variable nodes memories (M_{cv}, M_{vc}), respectively. The CNPs read data from M_{vc} in their required order and after processing write back in M_{cv} in the order required by VNPs, and vice versa for VNPs. The complicated intertask data dependencies result in complex memory accesses and difficult-to-resolve memory conflicts in the corresponding partially parallel architectures. In many practical MP algorithms, the variable node computations are implemented as additions of the variable node inputs and the check node computations as log or tanh function computation for each check node input and addition of the results of the log/tanh computations. In some simplified practical algorithms, the check nodes just compare their inputs to find the lowest and second lowest value. Since each node receives several inputs, the basic operations performed in nodes are the multi-input additions or multi-input comparisons.

The Tanner graphs corresponding to practical LDPC codes of the newest communication system standards involve hundreds of variable and check nodes, and even more edges. Thus, the LDPC decoding for these standards represents a massive computation, as well as complex storage and communication task. Moreover, as explained in the introduction, for realization of the multi-Gbps throughput required by these standards, massively parallel hardware multiprocessors are necessary. For such multiprocessors, the memory and communication architecture design plays a decisive role. To adequately support the design process for such applications, we proposed the quality-driven model-based design methodology [6] briefly discussed below.

3. Quality-Driven Model-Based Accelerator Design Methodology for Highly Demanding Applications

Our accelerator design method is based on the quality-driven design paradigm [32]. According to this paradigm, system design is actually about a definition of the required quality, in the sense of a satisfactory answer to the following two questions: what quality is required and how can it be achieved? To bring the quality-driven design into effect, quality has to be modeled, measured, and compared. In our

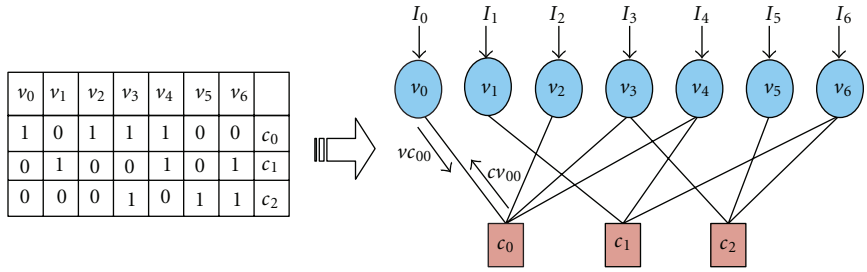

Figure 4: PCM for an (7,4) LDPC code and its corresponding Tanner graph.

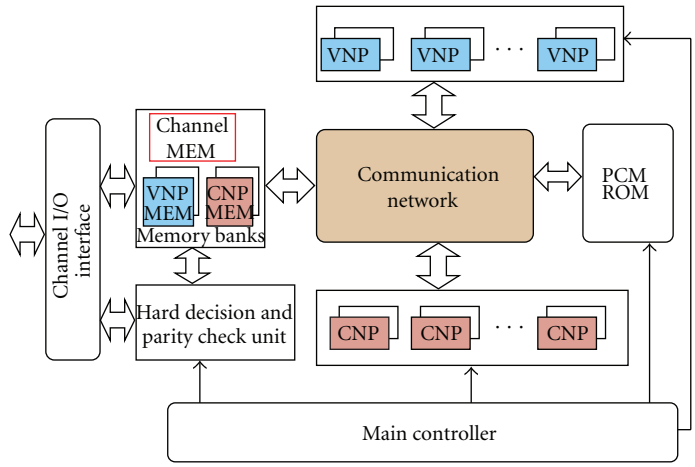

Figure 5: Example of a generic architecture template for LDPC decoding accelerators.

approach, the quality of the accelerator required is modeled in the form of the demanded accelerator behavior and structural and parametric constraints and objectives to be satisfied by its design, as described in [6]. Our approach exploits the concept of a predesigned generic architecture platform, which is modeled as an abstract generic architecture template (e.g., Figure 5). Based on the analysis results of the so modeled required quality, the generic architecture template is instantiated and used to perform the DSE that aims at the construction of one or several most promising accelerator architectures supporting the required behavior and satisfying the demanded constraints and objectives. This is performed through analysis of various architectural choices and tradeoffs. Our approach considers the macroarchitecture and microarchitecture synthesis and optimization, as well as the computing, memory, and communication structures' synthesis as one coherent accelerator architecture synthesis and optimization task, and not as several separate tasks, as in the state-of-the-art methods. This allows for an adequate resolution of the strong interrelationships between the micro- and macroarchitecture and computation unit, memory, and communication organization. It also supports an effective tradeoff exploitation between the micro- and macroarchitecture, the memory and communication architecture, and between the various aspects of accelerator's effectiveness and efficiency. According to our knowledge, the so formulated accelerator design problem is not yet explored in any of the previous works related to hardware accelerator design.

In more precise terms, our *quality-driven model-based accelerator architecture design method* involves the following core activities:

(i) *design of a pool of generic architecture platforms and their main modules, and platform modeling in the form of an abstract architecture template* (once for an application class),

(ii) *abstract requirement modeling* (for each particular application),

(iii) *generic architecture template and module instantiation* (for each particular application),

(iv) *computation scheduling and mapping on the generic architecture template instance* (for each particular application and template instance),

(v) *architecture analysis, characterization, evaluation, and selection* (for each constructed architecture),

(vi) *architecture refinement and optimization* (processing, interfacing, and memories abstraction refinement and optimization, for the selected architectures only).

The exploration of promising architecture designs is performed as follows (see Figure 6). For a given class of applications, a pool of generic architecture templates, including their corresponding processing units, memory units, and

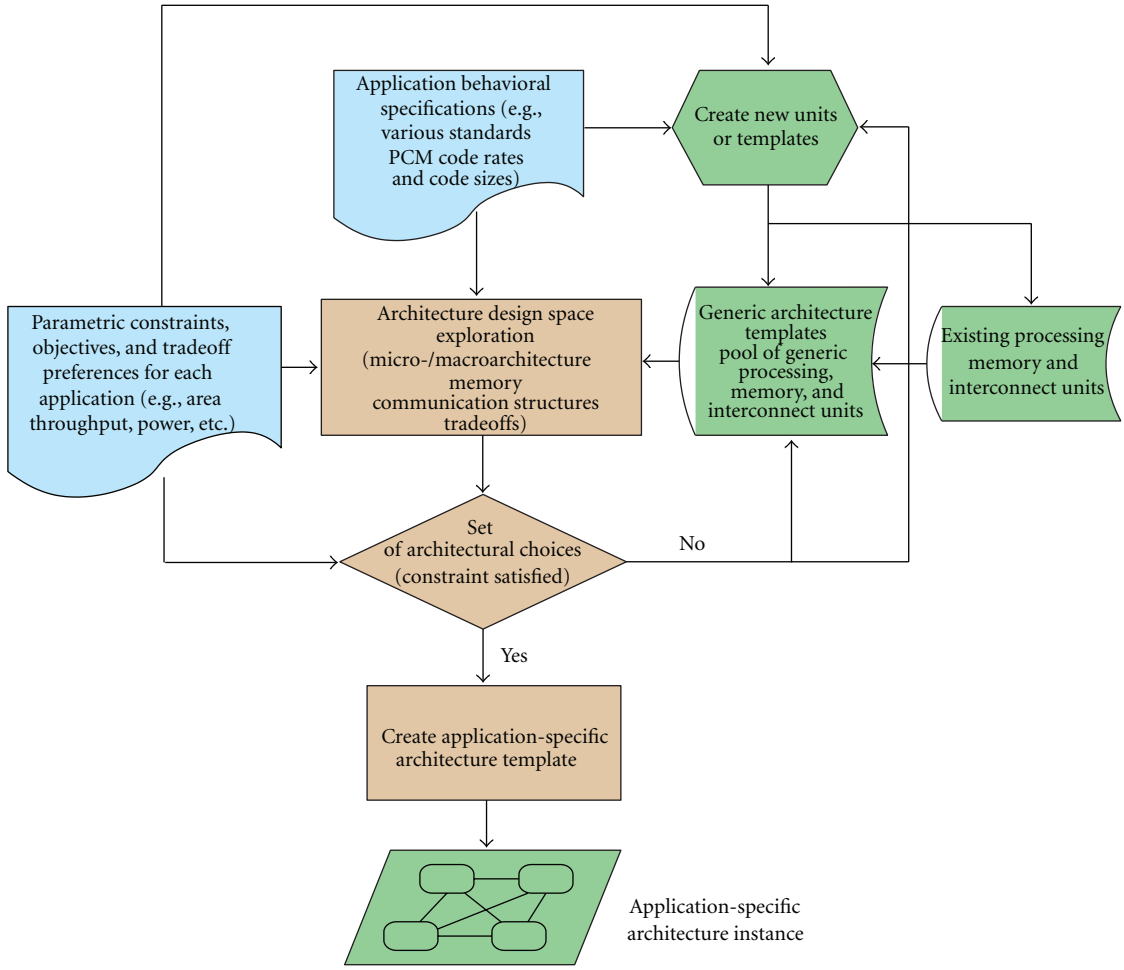

Figure 6: Architecture exploration framework of the accelerator design methodology.

other architectural resources, is prepared in advance by analyzing various applications of this class, and particularly, analyzing the applications' required behavior and ranges of their structural and parametric demands. Each generic architecture template specifies several general aspects of the modeled architecture set, such as presence of certain module types and the possibilities of the modules' structural composition and leaves other aspects (e.g., the number of modules of each type or their specific structural composition) to be derived through the DSE in which a template is adapted for a particular application. In fact, the generic templates represent generic conceptual architecture designs which become actual designs after further template instantiation, refinement, and optimization for a particular application. The adaptation of a generic architecture template to a particular application with its particular set of behavioral and other requirements consists of the DSE through performing the most promising instantiations of the most promising generic templates and their resources to implement the required behavior, when satisfying the remaining application requirements. In result, several most promising architectures are designed and selected that match the requirements of the application under consideration to a satisfactory degree.

Our architecture DSE and synthesis algorithm takes as its input the required accelerator quality (see Figure 7). The required accelerator quality is represented by the accelerator behavioral specification in a parallel form, the required accelerator throughput and frequency, and the required tradeoff between the accelerator area and power consumption, as well as the structural requirement to be constructed as one of the possible instances of the generic architecture template and its modules. In a large majority of practical cases, the throughput and clock speed are the hard constraints that must be satisfied, while the area, power, and their mutual tradeoffs are considered as the design objectives that have to be optimized. In these cases, the effectiveness of an accelerator is represented by the throughput and frequency constraints, while the area, power, and their mutual tradeoffs reflect its efficiency. This way the required accelerator quality is modeled, and this quality model is used to drive the overall architecture exploration and synthesis process that carefully stepwise constructs the most promising architectures. It is performed in the following three stages, each corresponding to one of the main design issues (subproblems) that have to be solved to result in complete accelerator architecture:

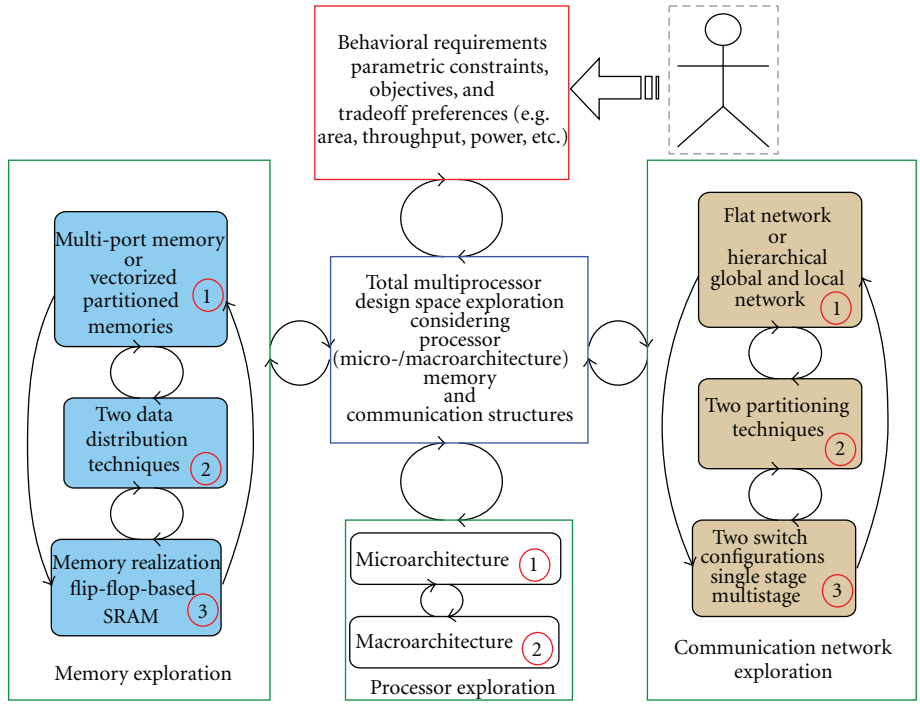

Figure 7: Design space exploration (DSE) of communication and memory architectures using various strategies.

(1) decision of the processor's micro- and macroarchitectures (processing parallelism) for each different data-parallel task,

(2) decision of the memory and communication architecture for selected micro-/macroarchitecture combinations,

(3) selection and actual composition of the final complete accelerator architecture.

Since the throughput and clock speed are the hard constraints and their satisfaction mainly depends on the processing parallelism, and in turn, the required processing parallelism decides to a high degree the memory and communication architecture, the architecture exploration starts with the decision of the processing parallelism (stage 1). In this stage, two major aspects of the accelerator design, being its microarchitecture and macroarchitecture, are considered and decided, as well as the tradeoffs between these two aspects in relation to the design quality metrics (such as throughput, area, energy consumed, cost, etc.). It is important to stress that these macro- and microarchitecture decisions are taken in combination, because both the macro- and microarchitecture decisions influence the throughput, area, and other important parameters, but they do it in different ways and to different degrees. For instance, by a limited area, one can use more elementary accelerators, but with less parallel processing and related hardware in each of them, or vice versa, and this can result in a different throughput and different values of other parameters for each of the alternatives.

In the second stage, the memory and communication architectures are decided for each of the step one constructed and selected candidate partial architectures representing particular micro- and macroarchitecture combinations (P_{mic}, P_{mac}). It is assumed that the storage and data transfer bandwidth per clock cycle must match the processing bandwidth, that is, bandwidth/cc $= P_{\text{mic}} \times P_{\text{mac}} \times b$, where P_{mic} and P_{mac} represent the data parallelism of the micro- and macroarchitecture for a given task, correspondingly, and b represents the bit width of data. To ensure the storage and data transfer bandwidth required by processors on a low cost and satisfactory delays, different memory and communication architectures are considered during the DSE. The DSE algorithm explores and selects the most promising of the memory and communication architectures for a particular micro-/macroarchitecture combination while taking into account the design constraints and optimization objectives. The memory and communication architectures and their exploration and synthesis strategies for a particular application being the main subject of this paper will be discussed in detail in the next section.

Finally, to decide the most suitable architecture, the most promising architectures constructed during the DSE are analyzed in relation to the quality metrics of interest and basic controllable system attributes affecting them (e.g., number of accelerator modules of each kind, clock frequency of each module, communication structures between modules, schedule, and binding of the required behavior to the modules, etc.), and the results of this analysis are compared to the design constraints and optimization objectives. This way the designer receives feedback, composed of a set of instantiated architectures and important characteristics of each of the architectures, showing to what degree the particular design objectives and constraints are satisfied by each of them. If

some of the constraints cannot be satisfied for a particular application through instantiation of given templates and their modules, new more effective modules or templates can be designed to satisfy the stringent requirements, or the requirements can be reconsidered and possibly lowered. Subsequently, the next iteration of the DSE can be started. If all the constraints and objectives are met to a satisfactory degree, the corresponding final application-specific architecture template is instantiated, further analyzed, and refined to represent the actual detailed design of the required accelerator.

4. Communication and Memory Architecture Design for High-End Multiprocessors

In this section, we propose some communication and memory design strategies that enable us construct effective and efficient architectures for the multi-processor accelerators. We then discuss how these strategies are incorporated in our architecture exploration framework, and how they are used to quickly explore the various tradeoffs among the different architecture options and to select the most promising architecture. Finally, we propose the memory exploration and synthesis techniques to ensure the required memory bandwidth in the presence of complex interrelationship between data and computing operations.

Our approach is based on the exploration of computation and communication hierarchies and flows present in a given application, and on using the knowledge from this exploration for the automatic design of communication and memory architectures. Based on the analysis of the communication hierarchies and flows, the processing elements are organized in a corresponding hierarchical way into several tiles (groups). The tiles are then structured into one global cluster or several global communication-free smaller clusters (if possible), and their respective data in memory tiles. The tiles and clusters replace a fully flat communication network with several much smaller hierarchically organized autonomous communication networks.

Since the global communication complexity and delays grow drastically with the increase of parallelism, we developed some strategies to decompose the global cluster into multiple much smaller global communication-free clusters. For a particular application, this partitioning is performed by taking into account the application parallelism and by adequate mapping of computation tasks and their data to the processors and memories, respectively. This localization of communication involving several small size clusters eliminates the global intertile communication and results in a substantial improvement of the communication architecture scalability for the highly demanding applications.

Secondly, in the cases the intertile global communication is unavoidable, we use a decomposition strategy in which we decompose one global cluster (global switch) into multiple smaller clusters (switches) again by exploiting a careful analysis of data in memories. Finally, we also exploit several different kinds of switches (e.g., single-stage switches or multistage switches), each appropriate to be used in a different context. All these strategies combined in a proper way result in resolution of the communication bottlenecks and related physical interconnect issues in the architecture. This way an optimized well-scalable communication architecture is designed, while at the same time realizing an effective and efficient application-specific memory-processor communication, as well as an adequate task and data mapping to particular processing elements and memories, respectively. The above-introduced strategies can be applied in different possible combinations. For example, a two-level hierarchical organization may be followed by partitioning or realized as the two-level network with different single-/multistage switch configurations. Different strategies combinations result in different tradeoffs. The above strategies and the order in which they can be applied are represented in the form of a flow diagram in Figure 7.

Due to the complex interrelationships between the data and computing operations at the task level and complex intertask data dependencies, an adequate customization of memory architecture is one of the major design tasks for the massively parallel hardware multiprocessors. For a given application, all data (input and intermediate) specified in the form of single and multi-dimensional arrays have to be stored in multiple shared memories. Different tasks and their corresponding processors impose different access requirements (read/write orders) on the shared memories. Taking into account the single task access requirements on the shared memories would certainly paralyze the other tasks that access the same shared memories for other computations. To ensure the required memory bandwidth and conflict-free data access, data have to be partitioned, distributed, and mapped in multiple vector or multibank memories, as discussed in Section 2. This way the overall complexity of the memory architecture will be lower and at the same time would satisfy the required memory bandwidth. The problems of data organization into vectors and the required number of shared vector memory tiles (partitions) are resolved together with the communication architecture design, when the flat communication network is transformed into the hierarchical network. However, providing as many shared vector memory tiles (partitions) as the processing tiles would only partially solve the problem due to the possible memory access conflicts. Therefore, the data distribution and data mapping in the partitioned memories are performed with the aim to eliminate the memory access conflicts, as well as to ensure that our communication strategies would be applicable. It is worth to be noted that our memory partitioning and data distribution approach avoid data duplication. The data distribution and data mapping approach are described below using an example of two heterogeneous data-parallel tasks sharing multiple memories.

Let us assume a set of m data-parallel tasks $T_i = \{T_1, \ldots, T_m\}$ and another set of n data-parallel tasks $T_j = \{T_1, \ldots, T_n\}$. Let $|P_i(P_{\text{mic}}, P_{\text{mac}})|$ and $|P_j(P_{\text{mic}}, P_{\text{mac}})|$ be the number of processing tiles allocated to the tasks T_i and T_j, respectively, where P_{mic} represents the microarchitecture parallelism, and P_{mac} represents the macroarchitecture parallelism of each processing tile. Let $|M_{i,j}| = P_i(P_{\text{mic}}) \times P_i(P_{\text{mac}})$ and $|M_{j,i}| = P_j(P_{\text{mic}}) \times P_j(P_{\text{mac}})$ be the number of memory tiles shared among the processing tiles $|P_i|$ and $|P_j|$. Further,

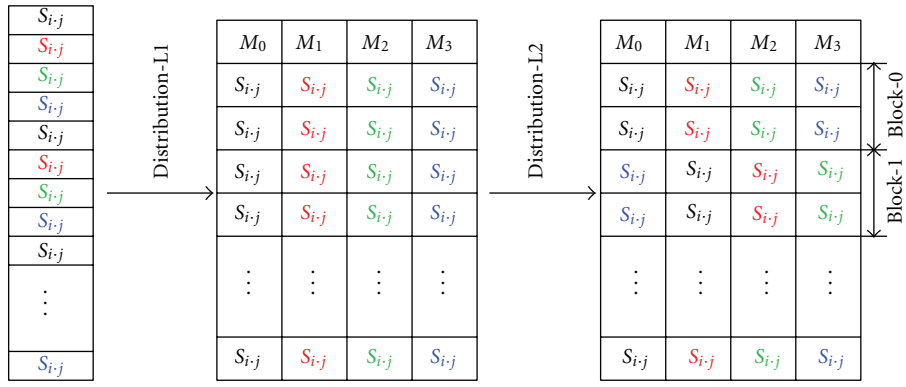

Figure 8: Data distribution strategies in multiple shared memories for conflict-free accesses.

we assume that $|P_i|$ reads data from $|M_{i,j}|$ and writes to $|M_{j,i}|$ and vice versa for $|P_j|$. For data distribution, we propose an interleaved (cyclic) data distribution scheme. This approach regularly and uniformly distributes data in memories, which enables us to use our communication strategies. Further, this approach has the additional benefit that it minimizes the complexity of the addressing logic. We perform data distribution based on interleaving in two stages, to resolve the read and write access conflicts, respectively. Depending on the number of shared memory tiles (partitions), the data distribution is performed as given by the equation below:

$$M_{i,j}(x) = S_{i,j} \% \left| M_{i,j} \right|, \quad \text{where } 0 \le x \le \left| M_{i,j} \right|, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \tag{1}$$

where $M_{i,j}(x)$ represents the specific shared vector memory tile to which a particular data tile $S_{i,j}$ is mapped, where the subscripts i and j in $S_{i,j}$ represent the data dependence between the task T_i and T_j, and $|M_{i,j}|$ represents the total number of shared memory tiles from which processors $|P_i|$ read and $|P_j|$ write their data. All the data tiles $S_{i,j}$ are organized as two-dimensional arrays that facilitate the automatic data distribution in the shared memory tiles $|M_{i,j}|$ using (1). Figure 8 shows our data distribution approaches in the shared partition memories with 4 memory partitions. This data distribution (distribution-L1) will resolve all the memory read conflicts for processors $|P_i|$ that will be in the case if no memory partitioning is done and all data is stored in a single memory (single port), as shown in Figure 8. On the other hand, when the processor tiles $|P_j|$ write to the share memory tiles $|M_{i,j}|$, it might result in write conflicts because of the order imposed by the $|P_i|$ processor tiles for conflict-free read on the data tiles, as given in (1). Therefore, we use another level of data interleaving so that the processor tiles $|P_j|$ write their data without any conflict, while ensuring that $|P_i|$ read accesses will not be effected. Unlike the interleaving which is at the level of a data tile, we perform this rather at the block level. All the data tiles distributed in the partitioned memories $|M_{i,j}|$ for the task $|T_i|$ are first divided into sets of equal size blocks (each block consists of a set of data tiles), then the data tiles of each block are skewed (interleaved) by a certain value. The data blocks are formed by taking into account the information about the set of tasks T_j and their relevant data tiles $S_{i,j}$ that are scheduled simultaneously on the processor tiles P_j. The block is formed in such a way that each block contains a single data tile $S_{i,j}$ from the scheduled subsets of tasks T_j, and to avoid the conflicts, data tiles are then interleaved (skewed) in each block by some value. This way the processor tiles $|P_j|$ can write to the shared memory tiles $|M_{i,j}|$ without any conflict, when ensuring that the corresponding read will not be effected. We can determine the block-level data distribution using the equation below:

$$B_n(x) = n, \quad \text{where } 0 \le n \le |B_n|, \tag{2}$$

where $B(x)$ represents the block index number, n represents the value of the interleaving (skew factor), and $|B_n|$ represents the total number of blocks. The same conflict-free read/write access order is valid for $|M_{j,i}|$ shared memory tiles except that the read/write access order is just reversed. It is equally possible that during data distribution for resolving the read conflicts, it might also resolve the write conflicts. In such scenario, the second level data distribution would not be needed. Further, the shared partitioned memories can be implemented using flip-flop- (FF-) based registers or embedded SRAM memories. We integrated into our DSE framework, the HP CACTI, a cache, and SRAM compiler, for memory characterization with different configurations required during DSE. The above strategies and the order in which they can be applied are represented in the form of a flow diagram in Figure 7. We will further explain the above-discussed communication and memory design approach and its strategies using as a representative test case the design of LDPC decoders for the future communication system standards.

4.1. Case Study: Communication and Memory Architecture Design for LDPC Decoders. Practical LDPC codes, such as those adopted in the IEEE 802.15.3c standards for future demanding communication systems, exhibit a very complicated, but not fully random, information flow structure, in which certain regularity and hierarchies are present [3]. According to our communication and memory architecture synthesis method introduced in the previous section, the

Table 1: Block-structured PCM, H_{base}, of 1/2 rate IEEE 802.15.3c LDPC code with 32 macrocolumns and 16 macrorows, size of each submatrix is 21×21, and codelength is 672; "—" represents zero matrices.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	—	—	—	5	—	18	—	—	—	—	3	—	10	—	—	—	—	—	—	5	—	—	—	—	—	—	—	5	—	7	—	—
2	0	—	—	—	—	—	16	—	—	—	—	6	—	—	—	0	—	7	—	—	—	—	—	—	—	10	—	—	—	—	—	19
3	—	—	6	—	7	—	—	—	—	2	—	—	—	—	9	—	20	—	—	—	—	—	—	—	—	—	19	—	10	—	—	—
4	—	18	—	—	—	—	—	0	10	—	—	—	—	16	—	—	—	—	9	—	—	—	—	—	4	—	—	—	—	—	17	—
5	5	—	—	—	—	—	18	—	—	—	—	3	—	10	—	—	5	—	—	—	—	—	—	—	—	—	—	—	—	—	7	—
6	—	0	—	—	—	—	—	16	6	—	—	—	0	—	—	—	—	—	7	—	—	—	—	—	—	—	—	19	—	—	—
7	—	—	—	6	—	7	—	—	—	—	2	—	—	—	—	9	—	20	—	—	—	—	—	—	—	—	—	—	—	10	—	—
8	—	—	18	—	0	—	—	—	—	10	—	—	—	—	16	—	—	—	—	9	—	—	—	—	—	—	—	—	—	—	—	17
9	—	5	—	—	—	—	—	18	3	—	—	—	—	—	10	—	—	5	—	—	4	—	—	—	—	5	—	—	—	—	—	7
10	—	—	0	—	16	—	—	—	—	6	—	—	—	0	—	—	—	—	—	7	—	4	—	—	—	—	—	10	—	19	—	—
11	6	—	—	—	—	—	7	—	—	—	—	2	9	—	—	—	—	—	20	—	—	—	4	—	19	—	—	—	—	—	10	—
12	—	—	—	18	—	0	—	—	—	—	10	—	—	—	—	16	9	—	—	—	—	—	—	12	—	—	4	—	17	—	—	—
13	—	—	5	—	18	—	—	—	—	3	—	—	—	—	—	10	—	—	5	—	—	—	—	—	—	—	5	—	—	—	—	—
14	—	—	—	0	—	16	—	—	—	—	6	—	—	—	0	—	7	—	—	—	—	—	—	—	10	—	—	—	—	—	—	—
15	—	6	—	—	—	—	—	7	2	—	—	—	—	9	—	—	—	—	—	20	—	—	—	—	—	19	—	—	—	—	—	—
16	18	—	—	—	—	—	0	—	—	—	—	10	16	—	—	—	—	9	—	—	—	—	—	—	—	—	—	4	—	—	—	—

information flow structure of such an application has to be carefully analyzed. The aim of this analysis is to discover the application regularities and hierarchies in order to exploit them for the design of an effective and efficient communication architecture (possibly several levels) of hierarchical localized communication clusters. For instance, the practical LDPC codes are defined by block-structured PCMs. A block-structured PCM groups a certain number of rows (CNs) of PCM into a macro-row and the same number of columns (VNs) into a macro-column, creating this way the corresponding macroentries of the block matrix. For example, 21 rows and columns form a macro-row and macro-column, respectively, for the PCM shown in Table 1. The particular macro-entries of this table represent particular submatrices corresponding to the particular 21 rows and 21 columns.

The interconnections among particular macrorows and macrocolumns of the block-structured PCM are defined by the nonzero entries (submatrices), zero entry "—" means no interconnection. Every macro-row is connected to a different subset of macrocolumns in a complex pseudorandom way and vice versa. For example, the macro-row {1} is connected to the macrocolumns {4, 6, 11, 13, 20, 28, 30}, and the macro-column {1} is connected to the macrorows {2, 5, 11, 16}. However, the interconnections within each submatrix of the block-structured PCM are defined by regular circularly shifted identity matrices with shift values represented by the nonzero entry in the matrix. Hence, in the corresponding hardware multi-processor, the communication within a single nonzero sub-matrix can be realized locally using a quite regular local communication network, while the communication among the macrorows and macrocolumns is irregular and can be realized using a global communication network, as shown in Figure 9. This substantially decreases the communication network complexity (see Figure 10(b)) compared to the case of the flat communication scheme (see Figure 10(a)) for different micro-/macroparallelism combinations. In these and the following figures presenting experimental results, $P(a, b)$ denotes a combined micro- and macroarchitecture parallelism. In tuple $P(a, b)$, a represents the microarchitecture parallelism of a processor (i.e., the number of processor inputs/outputs), and b represents the macroarchitecture parallelism (i.e., the number of processors). The tuple $P(a, b)$ represents a certain micro- and macroarchitecture combination with the combined micro- and macro-parallelism (a, b) of the CNP processors, correspondingly (shown on the x-axis in the figures presenting the results). Similar notation for the combined processing parallelism is used for the VNP processors (although, not shown on the x-axis of the result figures). As shown in Figure 11(a), the area saving is as high as *25 times* for the architecture instance $P(4, 336)$. Similarly, except for the low parallelism levels for which the flat scheme performs well, for the moderate and high level of parallelism, the hierarchical two-level interconnect approach provides superior performance. The performance gain is as high as *12 times* for architecture instance $P(2, 336)$, as shown in Figure 11(b). Moreover, the performance saturates at a certain higher parallelism level for the flat communication scheme, and a drop in performance can be observed by further increase in parallelism, because the switch delays dominate the processor delays. The same trend can be observed for the two-level communication network, but at a different parallelism level (e.g., $P(4, 336)$, $P(8, 84)$), as shown in Figure 10(b).

Our area estimates are very accurate as we perform a prior floor planning of the top-level design (macroarchitecture) and the actual design and physical characterization of various instances of the generic architecture modules (processors, memories, and communication resources), when accounting for the interconnect effects during the module characterization. Since the macroarchitecture design (composition of architecture modules to form the accelerator) is very regular and follows the same general structure for

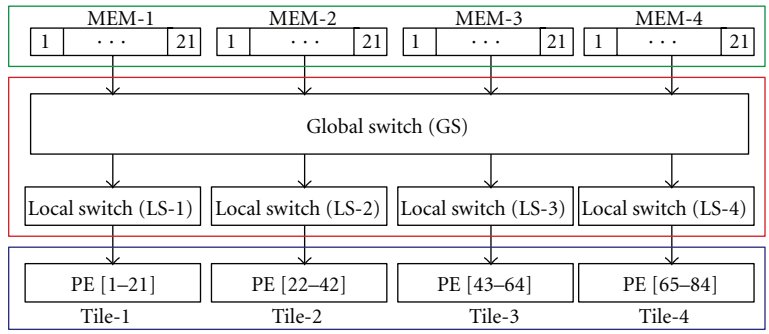

Figure 9: Example of the hierarchical communication network of LDPC decoders for IEEE 802.15.3c LDPC code decoder of 1/2 rate (R), code length 672 (L), and (micro, macro) parallelism of (1,84).

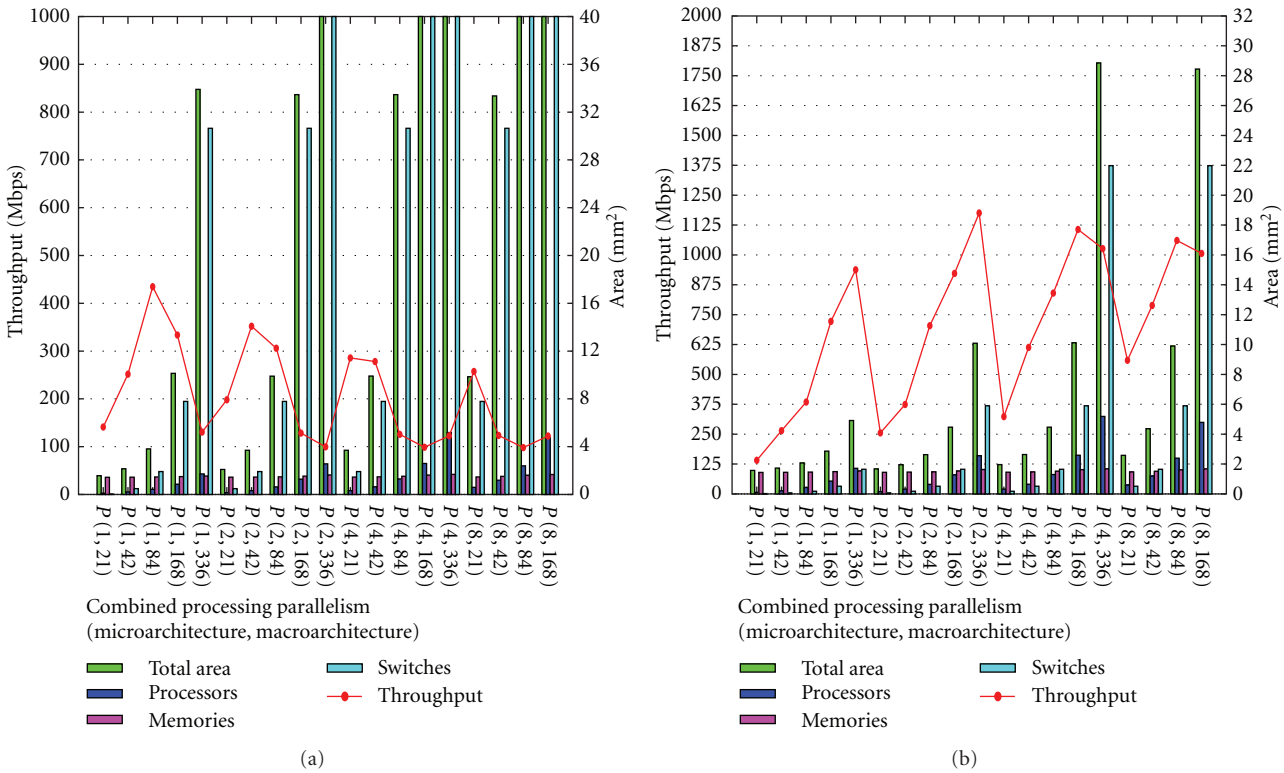

Figure 10: Area/performance tradeoffs for the flat communication network are shown on the (a) and for the two-level hierarchical on (b).

all architecture instances, the corresponding floorplan and actual layout are very regular and have almost the same general form for all architecture instances. Therefore, the parameter predictions based on the parameter values for the individual blocks and the floorplan do not much differ from the actual values from the layout both regarding the area and performance estimates. The blocks and the top-level design are modeled in Verilog HDL that can be targeted to various implementation technologies. For performing the experiments reported in this paper, it has been targeted at CMOS 90 nm technology (TSMC 90 nm LPHP standard Cell Library). For blocks characterization (parameters estimations), Cadence Encounter RTL compiler was used for synthesis and Cadence Encounter RTL-to-GDSII system 9.12 for physical place and route. The area, delay, and computation clock cycles estimates of both the CNP and VNP processors with various microarchitecture parallelisms are provided in Table 2. To compute the total area of several processors, the total processors' area is calculated using simple addition of the area of individual processors. For instance, for the tuple $P(1, 84)$, that is, 84 serial processors, the total processors' area is 0.508116 mm^2 (=$84 \times A_{\text{cnp}} + 84 \times A_{\text{vnp}}$), where A_{cnp} and A_{vnp} represent the area of CNP and VNP processors each with the microarchitecture parallelism of 1.

Moreover, we also observe that the communication network and memory dominate the processors area, as shown in Figure 10. For instance, for the tuple $P(1, 84)$, the communication network's area is *4.5 times* and the memory's area

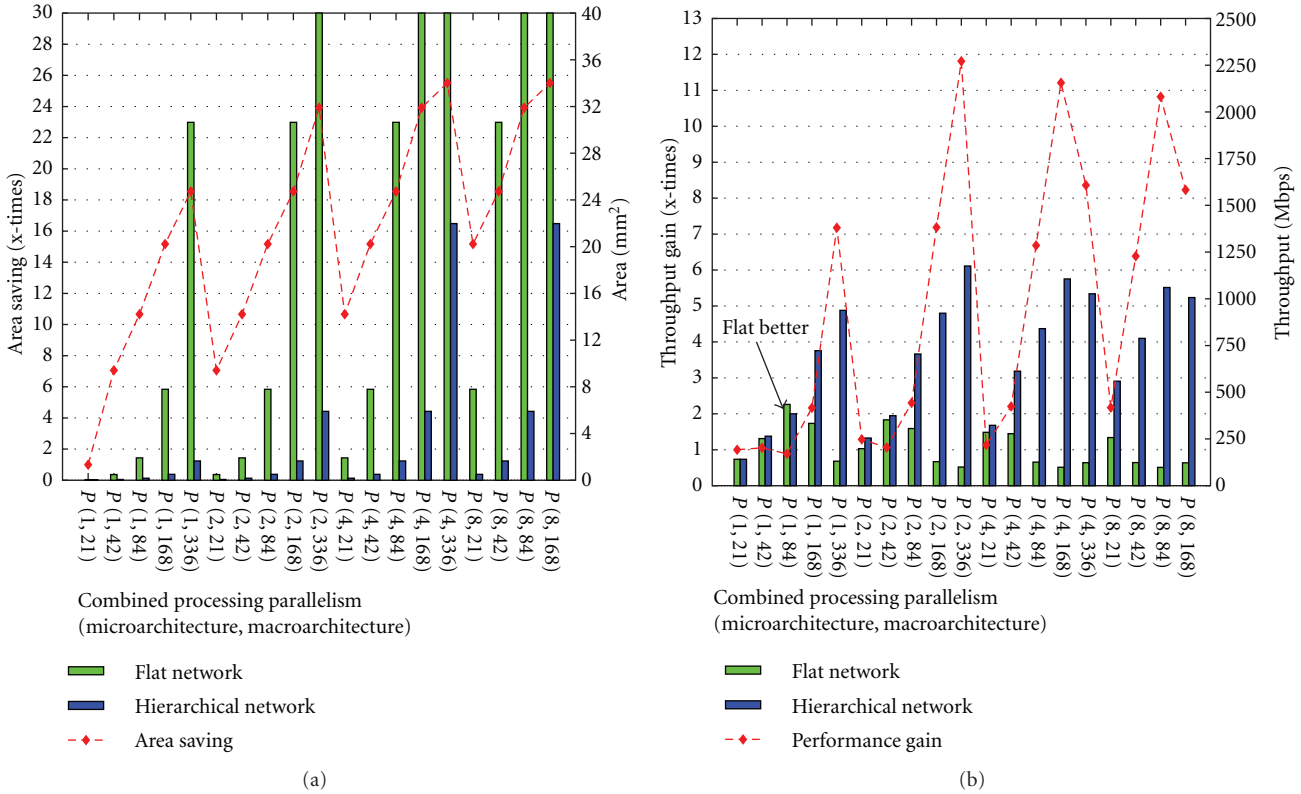

Figure 11: Area tradeoffs for the flat versus hierarchical communication network are shown on the (a) and throughput tradeoffs on (b).

Table 2: Characterization results for CNP and VNP processors using TSMC 90 nm LPHP standard cell library.

Processor type	Parameters	Microarchitecture parallelism			
		1	2	4	8
CNP	Area (mm^2)	0.002759	0.003933	0.005998	0.008709
	Delay (ns)	0.751	1.413	2.105	2.709
	Clock cycles	8	4	2	1
VNP	Area (mm^2)	0.003290	0.005089	0.011673	—
	Delay (ns)	0.847	0.921	1.366	—
	Clock cycles	4	2	1	—

3.4 times larger than the processors's area, respectively, as shown in Figure 10(a). In particular, for the higher processing parallelism level, the communication network influence on the area much dominates the processor influence (see Figure 10). The processor's contribution to the total area is shown in the dark blue color, communication network' contribution in light blue, and memory's contribution in magenta color in Figure 10.

The throughput of an LDPC decoder can be estimated analytically based on the two-phase message passing (TPMP) decoding algorithm using the following formula:

$$T_{\text{Mbps}} = \frac{R \times N \times F_{\text{MHz}}}{\text{CC}/I \times I_{\text{tot}}}, \quad (3)$$

where T_{Mbps} stands for the throughput in Mbps, R stands for the code rate, N stands for the code length (size of data frame), I_{tot} stands for the total number of iterations required to decode a code word, F_{MHz} stands for the clock frequency, and CC/I is the clock cycles required for a single iteration, that is, the schedule length in CC when multiplied with I_{tot}. For a particular LDPC code N, R and I_{tot} are decided in advance for a particular application and its frame error rate (FER). Therefore, the parameters that remain to determine the throughput are the CC/I and F_{MHz}. The CC/I depends on the micro- and macroarchitecture parallelism exploited. The clock speed F_{MHz} depends on the processor's critical path delays plus the physical delays of the communication and memory structures. The CC/I is directly influenced by the processor micro- and macroarchitecture parallelism. For instance, a fully parallel processing element would perform the computation in a single cycle, while the serial will take as many clock cycles as the total number of inputs of a given multi-input operation (see Table 2). The throughput for each micro- and macroarchitecture combination is estimated

based on the actual performance (clock speed and clock cycles), area, and power numbers corresponding to particular processor parallelism from processor characterization. Based on these actual performance numbers, (3) is then used to compute the final throughput in Mbps.

To be able to obtain the experimental results presented in this section, we had to synthesize and analyze a large set of promising hardware multi-processor architectures, as shown in Figure 10. The synthesis and evaluation of such a large set of architecture instances in a reasonably short time was only possible through usage of our automated DSE framework.

There are following four types of memories involved in the decoding of LDPC codes:

(i) M_{cv} to store the CN messages,

(ii) M_{vc} to store the VN messages,

(iii) M_{ch} to store the channel messages I_{ch},

(iv) M_{HD} to store the hard decision messages V_{HD}.

The check node memories M_{cv} and variable node memories M_{vc} are shared between the CNP and VNP processors. The CNP reads the check node data from the M_{cv} and after processing writes the result data to the M_{vc} memories. Similarly, the VNP reads the variable node data from the M_{vc} and after processing writes the result data to the M_{cv}. This represents back-to-back (cyclic) data dependencies between the CNPs and VNPs. The total amount of data required to be stored in M_{cv} and M_{vc} memory is equal to the number of nonzero (NZ_{sm}) elements (sub-matrices) of the block-structured PCM. Since the total number of nonzero elements in the rate 1/2 IEEE 802.15.3c 672-bit LDPC code is 108, 108 elements have to be stored in each of the M_{cv} and M_{vc} memories. Please note that a sub-matrix in the block-structured PCM represents the interconnections between the CNs and VNs and should not be considered as the actual data processed by the CNPs and VNPs. Moreover, for each type of task, the data related to each of the nonzero element (sub-matrix) of the block-structured PCM can be stored in a single vector location of a single vector memory, because the submatrices actually represent the identity-shifted matrices. In the identity matrices, there is only a single nonzero element in each row or column. The width, depth, and number (partitions) of each type of memory (M_{vc} or M_{cv}) depend on the processing parallelism exploited for each kind of processors (CNPs and VNPs). For instance, consider the case of the microarchitecture parallelism of four and the macro-architecture parallelism equal to the total sub-matrix parallelism (21 for rate 1/2 672-bit code). In this case, all the data for the CNPs and VNPs can be stored in four vector memories of each kind M_{cv} and M_{vc}. As the macroarchitecture parallelism is equal to the total sub-matrix parallelism, and for each sub-matrix, the related data can be stored in a single location of a single vector memory; a single vector memory is sufficient to provide the necessary bandwidth for each kind of processors (CNPs and VNPs). Four memory partitions are required as the assumed microarchitecture parallelism is four. This way the aggregate memory bandwidth is satisfied by four partitioned vector memories M_{cv} and M_{vc} for the CNP and VNP, respectively. Concerning the size of the memory partitions, each of the memory partitions is of depth $\lceil \text{NZ}_{\text{sm}}/(4 \times 1) \rceil$ and width of *sub-matrix* $\times b$ ($= 21 \times b$), where b represents the data width of CN and VN messages.

The problem yet to be solved is the data distribution and data mapping in the so constructed partitioned shared memories (M_{cv} and M_{vc}) located between the CNPs and VNPs. An adequate resolution of this problem is necessary due to different data access patterns of CNPs and VNPs to the shared memories. The data can be accessed element by element in case of the fully serial processors or at once in case of the fully parallel processors. A CNP requires to access the data from the same (macro-)row at the nonzero locations (row-major order), while a VNP requires to access the data from the same (macro-)column (column-major order). For instance, the CNPs of macro-row {1} need to access the data from the nonzero locations {4, 6, 11, 13, 20, 28, 30} of macro-row {1} stored in M_{cv} (see Table 1). While the VNPs of macro-column {1} require to access the data from the nonzero locations {2, 5, 11, 16} of macro-column {1} stored in M_{vc}. Both in the case of CNP and VNP, the processed data have to be stored back in M_{vc} and M_{cv} in the same access order as for read, respectively. Thus, the CNPs and VNPs share the M_{cv} and M_{vc} for read and write, but with a different access order. The CNP accesses the shared M_{cv} rowwise for read, while the VNP accesses the same shared memory columnwise for write. On the other hand, the VNP accesses the shared M_{vc} columnwise for read and the CNP accesses the same shared memory rowwise for write. Thus, in order to ensure that both the read and write accesses for both kinds of processors (CNPs, VNPs) would be conflict-free, data have to be appropriately distributed and mapped in the shared M_{vc} and M_{cv} memories. It is not possible to access the shared memories rowwise and columnwise without any access conflict.

Some approaches have been proposed in the literature to overcome the memory access conflicts for LDPC decoding [33–35]. These approaches use as many vector memories as the number of nonzero (NZ_{sm}) elements in the block-structured PCM for each task. However, this causes an increase in the number of memories, even for the proposed architectures which exploit a low processing parallelism. For the case of IEEE 802.15.3c codes, it would result in as high as 216 memories with a depth of one and width of $(21 \times b)$. Also, 32 memories for each of the M_{ch} and M_{HD} messages would be required. In our method, the memory partitioning is performed as discussed above for each architecture instance. The data distribution and related data mapping is performed using the approach described in the previous section. The data distribution and related mapping into the shared memories M_{cv} and M_{vc} and among the CNP and VNP tiles are performed taking into account the processing parallelism and the related task mapping. Our data partitioning, data distribution, and mapping approach provide as many memory partitions as the number of processing tiles. This enormously reduces the number of shared vector memories compared to the proposed approaches [33–35]. Since the M_{ch} and M_{HD} memories are only accessed by the VNPs, their memory partitioning, data distribution, and data mapping are trivial, as these memories are not shared among the different tasks.

Due to small sizes of data involved in the decoding of IEEE 802.15.3c LDPC codes, the influence of memories on the overall area and delay remains low in the whole range from centralized and relatively larger sizes to extremely distributed and very small sizes, as shown in Figure 10. Nevertheless, adequate memory design, as well as data distribution and mapping to distributed memories, is of primary importance for an effective and efficient communication design. In consequence, even for applications with small data sizes, the memory architecture design remains one of the major design issues.

Recently, several papers on architectures for processing a single sub-matrix of a single macro-row (serial CNP) and a single sub-matrix of a single macro-column (serial VNP) are published. However, processing of a single sub-matrix in isolation only requires a simple local communication network (switch) and a simple memory structure [22–26]. It does not solve the problem of an effective and efficient processing of the whole PCM matrix and related communication architecture for this aim. Some architectures for processing only a single macro-row and a single macro-column were also proposed, which required multiple local communication networks but no global communication network [27–29]. However, for the demanding accelerator cases, multiple macrorows and macrocolumns have to be processed in parallel, and this requires solution of a much more complex system of global and local communication problems. Our solution to these problems represents a generic hierarchical communication network and distributed memory architecture, as shown, for example, in Figure 9.

In this paper, we propose solutions for the actual total memory and communication problem, and not only for some of its isolated parts, as considered by the published research. Despite the local regularity at the level of a single submatrix, the global communication network complexity quite quickly increases with the increase of the micro-/macroparallelism (see Figure 10(b)). Therefore, to improve the scalability of the communication architectures for demanding applications, we proposed partitioning techniques which reduce the complexity of the global intertile communication. These techniques are discussed in the following sections.

4.2. Communication and Memory Partitioning Strategies

4.2.1. Communication Partitioning Based on Data Distribution in Memories. The basic idea of this technique is to reduce the data distribution related to a subset of node tiles from all memory tiles to a minimum possible number of shared memory tiles. This way, the corresponding subset of the processing tiles to which the node tiles are mapped would not be required to communicate with all the shared memory tiles. In consequence, a simpler communication network will be sufficient to communicate a specific subset of node tiles to a specific subset of shared memories tiles. This way, the single global switch can be partitioned into several smaller switches. A necessary and sufficient condition to be satisfied for this subset of node tiles sharing a subset of memory tiles is that this subset of nodes cannot be scheduled together (although they can be processed in parallel) due to the memory access conflict. For example, let us consider an accelerator instance with 2 tiles, each with a microparallelism of 2, and a particular memory organization with each node tile data stored in a vector word of a single-port vector memory or a bank of a multibank memory (see Figure 12). According to our data distribution and mapping methodology presented in Section 3, the data distribution and assignment to different shared memories is performed in the way to store all the related data in a minimum number of shared memory tiles, while ensuring that the data will be accessed without conflict. The corresponding memory assignment and data mapping for accelerator instance with 2 tiles each and with microparallelism of 2 is shown in Table 3. The subscripts a and b in $s_{a,b}$ represent the data related to a particular check node a and variable node b tiles, respectively. The node tiles {1,5}, {2,6}, {3,7}, and {4,8} can be scheduled together (one possible way) as the required data is located in separate memories. Moreover, all data related to variable nodes {1} and {5} are stored, respectively, in memories $\{M_0, M_3\}$ and $\{M_1, M_2\}$. This way, both tiles have to communicate with a subset of shared memories (only 2 in this case), and the corresponding single global intertile communication network is partitioned into two smaller communication networks. Unfortunately, this approach cannot be applied to the case of high parallelism levels. With the parallelism increase, data have to be distributed into more memory banks for parallel access. In such a scenario, the intertile communication is becoming unavoidable.

Table 3: Data distribution and assignment in the partitioned memory (M_{cv}) for an architecture instance with the combined processing parallelism $P(2, 42)$ both for the CNP and VNP processors.

	Check node memory banks (M_{cv})			
Word	M_0	M_1	M_2	M_3
w_0	$s_{1,4}$	$s_{1,6}$	$s_{2,1}$	$s_{2,7}$
w_1	$s_{3,3}$	$s_{3,5}$	$s_{4,2}$	$s_{4,8}$
—	—	—	—	—
w_8	$s_{5,1}$	$s_{5,7}$	$s_{6,2}$	$s_{6,8}$
w_9	$s_{7,4}$	$s_{7,6}$	$s_{8,3}$	$s_{8,5}$
—	—	—	—	—
w_{16}	$s_{9,2}$	$s_{9,8}$	$s_{10,3}$	$s_{10,5}$
w_{17}	$s_{11,1}$	$s_{11,7}$	$s_{12,4}$	$s_{12,6}$
—	—	—	—	—
w_{24}	$s_{13,3}$	$s_{13,5}$	$s_{14,4}$	$s_{14,6}$
w_{25}	$s_{15,2}$	$s_{15,8}$	$s_{16,1}$	$s_{16,7}$
—	—	—	—	—

4.2.2. Communication Partitioning Based on Data Identification. With increase of the number of tiles, it becomes impossible to avoid the intertile communication using the technique presented in the previous section. Data have to be distributed into multiple vector or multi-bank memories for conflict-free parallel access, and in consequence, a global interconnect network is required. However, the global switch complexity increases drastically with the parallelism increase from moderate to high (see Figure 10(b)). Therefore, rather

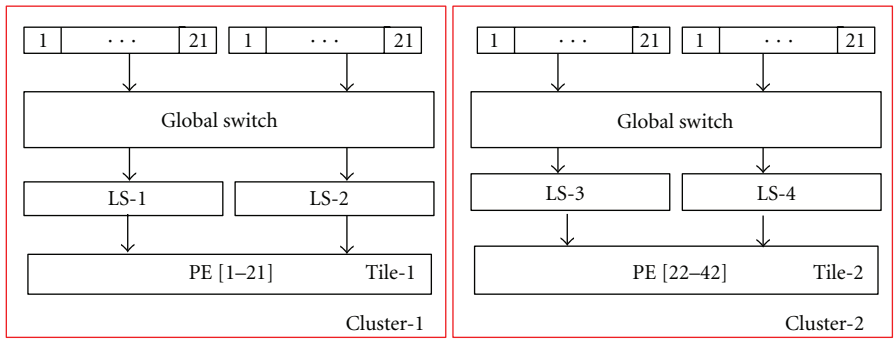

Figure 12: Data-distribution-based communication partitioning.

than using a single global switch to provide the intertile communication, different combinations of several switches and associated switch sizes should be used to reduce the complexity of the intertile communication. This decomposition of a single global switch into several much smaller ones can be achieved by taking advantage of the communication and computation regularity and hierarchy present in the application while considering the individual memory and processor tiles. As in the two-level hierarchical communication network, each one of the memory tiles communicates with one of the processor tiles at a time through a single global switch. Therefore, we have to explore the communication hierarchies and flows at the level of memory banks of a memory tile and at the level of processor of a processor tile. More specifically, by analysis of the specific characteristic of the memory-processor communication patterns, we identify specific subsets of banks in different memory tiles that simultaneously communicate with only specific corresponding subsets of processors in different processor tiles. In case of the LDPC codes from our case study, the memory banks $\{M_{1,1}, M_{2,1}, M_{3,1}, M_{4,1}\}$ in different memory tiles communicate with a subset of processors $\{P_{1,1}, P_{2,1}, P_{3,1}, P_{4,1}\}$ in different processor tiles, for the architecture instance shown in Figure 9. In $M_{a,b}$, a represents a memory number and b bank number, and similarly, in $P_{a,b}$, a represents a tile number and b processor number. Based on the identified patterns, a corresponding application-specific communication among the memories and processing tiles is then realized using several switches of much smaller sizes compared to the size of the single global switch. Both the number of switches and the size of each switch are decided by the processing parallelism. For the architecture instance shown in Figure 9, the required switch size is 4 (subset of 4 elements), while the required number of switches is 21 (21 subsets of 4 elements), as shown in Figure 13. Through this partitioning, the complexity of the single intertile global switch is much reduced, and the exploitation of the massive parallelism present in the application is possible at a reasonable cost. As we can see from the experimental results (see Figure 14(a)), much higher gain can be obtained in throughput due to the much shorter interconnect delays of the much smaller switches compared to the nonpartitioned global switch (see Figure 10(b)). The performance gain for all the accelerator instances as high as *5 times* on the cost of a very small area penalty of less than

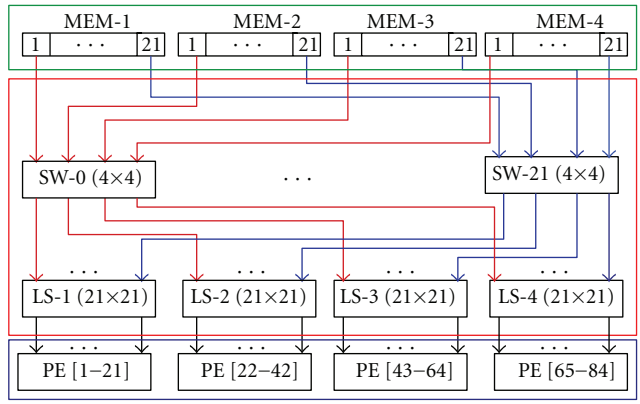

Figure 13: Data-identification-based communication partitioning.

1% is compared to the nonpartitioned two-level hierarchical communication network. Nevertheless, the communication network area still dominates the area of processing elements and memories. Therefore, we introduce one more technique to further reduce the communication network area, while preserving the performance.

4.2.3. Usage of the Single-Stage versus Multistage Switches. For each approach discussed so far, ranging from a flat to a partitioned hierarchical two-level communication network, we can further explore the design tradeoffs regarding communication architecture by using different kinds of physical switch networks (e.g., single-stage versus multistage switches). Moreover, for the two-level hierarchical communication network, the single-stage and multi-stage switches can also be used in combination. For example, we can utilize single stage switches for global interconnects and multi-stage switches for local interconnects or vice versa, and so forth. Moreover, we can apply this approach to the already partitioned single global switch using one of our earlier introduced partitioning approaches or to a nonpartitioned two-level hierarchical single-global-switch-based network. This enables us to explore different design tradeoffs regarding performance, area, and power consumption and ultimately ensure a good scalability. The single-stage and multi-stage switches are differently organized in terms of stages, which in turn results in different values for physical switch parameters, such

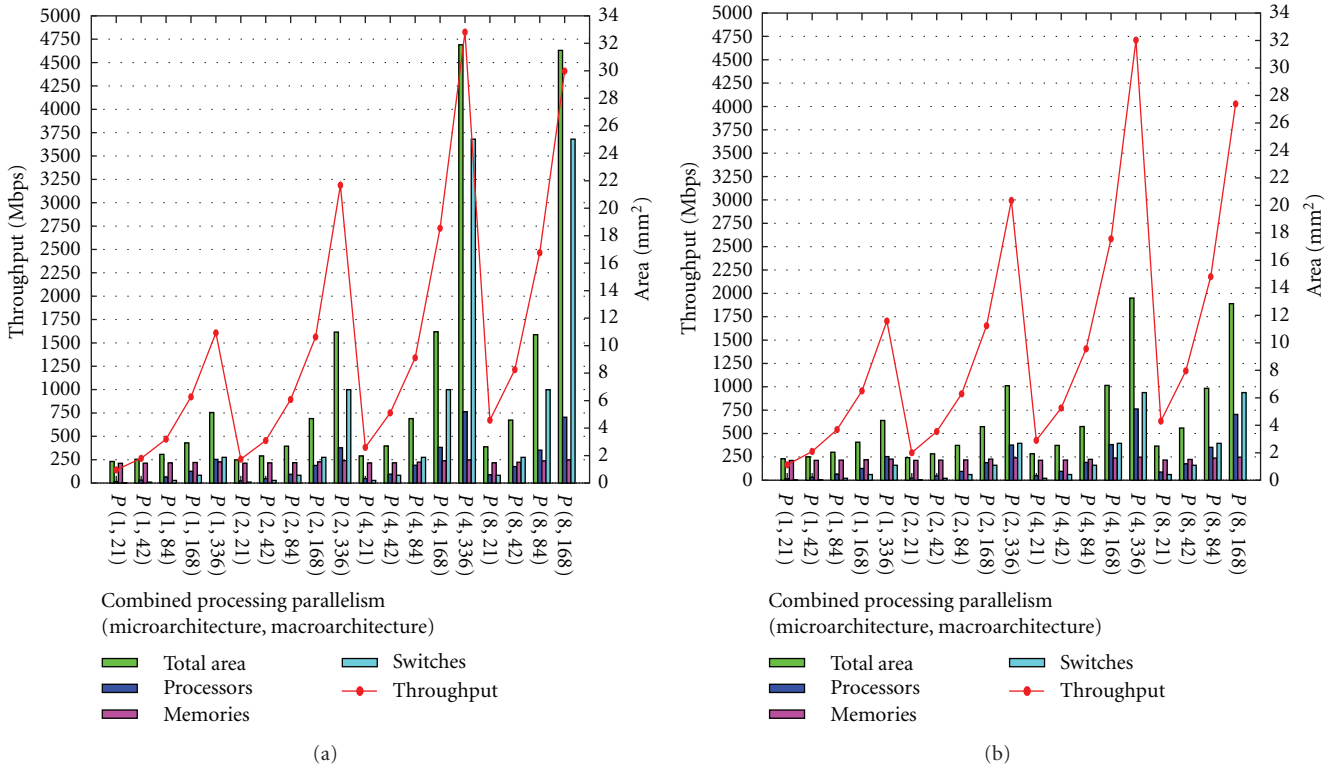

FIGURE 14: Area/performance tradeoffs for the data-identification-based communication partitioning shown at (a), and the same partitioned network when realized using multistage switches shown at (b).

as area and delay. For example, the local intra-tile communication network of LDPC decoders can be realized using a single-stage multi-input multiplexer- (MUX-) based shifters or by multiple simple 2×1 multiplexers cascaded in multiple stages (log shifters). Figure 14(b) shows the experimental results for the area and performance of a partitioned two-level communication network. The local and global switches are realized using multi-stage switches. The partitioned two-level multi-stage (both local and global) communication network outperforms in area the single-stage (both local and global) realization (see Figures 14(a) and 14(b)). There is a performance penalty of less than *1%* at most, while a significant area saving is obtained, as high as *4 times* compared to the case of a partitioned global communication network with single-stage switches. The communication network complexity is much reduced, which in turn ensures a better scalability for the massively parallel multi-processor accelerators.

4.3. Experimental Results Discussion and Comparison. We conducted a series of experiments for IEEE 802.15.3c LDPC codes with different micro-/macroarchitecture parallelism combinations to explore the numerous complex design tradeoffs, specifically, in relation to the memory and communication architecture. We performed our experiments for the 1/2 rate IEEE 802.15.3c LDPC code with code length 672, assuming a high data precision of 8 bits for communication and 10 iterations per frame. Most of the more specific results of our experiments are discussed in the previous sections to illustrate the memory and communication design approaches. However, the overall view and general conclusions are not less important. The fully flat communication scheme can only be used for the low-end applications, because its complexity explodes for the moderate and high parallelism levels, as shown in Figure 15. Moreover, for the moderate parallelism levels, the two-level hierarchical communication network performs well. It delivers large area savings and performance gains for all the accelerator instances, as high as *25 times* in area and as high as *12 times* in performance. Unfortunately, it shows nonscalable behavior for the massively parallel multi-processor cases, represented in green color in Figure 15. To ensure a much better scalability of the communication architectures for the high-end multiprocessors, our combined partitioning techniques are required. Although there is a performance gain for all the accelerator instances and as high as *5 times*, there is an area penalty of less than 1% compared to the non-partitioned two-level hierarchical communication network, represented in blue color in Figure 15. Therefore, multi-stage and single-stage switch combinations are adequately used in the implementation of the communication architectures to preserve the performance level while reducing the area. The multi-stage/single-stage switch combination approach results in a low performance penalty of less than 1% at most, while offering a significant area savings as high as *4 times*.

An interesting observation is that independent of the two-level hierarchical communication network partitioned

or not, the application of different combinations of single-stage and multi-stage realization to local and global switches always results in a superior scalability in one design dimension or in the other design dimension. Our experiments on different switch combinations show that the highest performance is achieved for the single-stage global switch and the multi-stage local switches, while for the lowest area, the multi-stage realizations are required for both the global and local switches. The same sort of behavior can be observed for the two-level nonpartitioned communication network, when the single and multi-stage switch combinations are applied. The results of our experiments show that a partitioned two-level communication network is the most promising one; it improves the scalability of communication architecture in all design dimensions with excellent tradeoffs. In general, for all the architectures having a low parallelism at both levels and for all communication architectures, both the performance and area scale linearly. This trend is clearly visible (see Figure 15) for the combined parallelism of up to 84 for architecture instances such as $\{P(1,84), P(2,42), P(4,21)\}$. For the higher parallelism levels beyond these points, the flat networks do not scale, but the simple hierarchical communication networks scale to some extent for the moderate parallelism levels, but much worse for slightly higher parallelism levels, and they saturate at the massively parallel micro-/macroarchitectures such as $P(4,168)$. The partitioning techniques with multi-stage and single-stage switch combinations are necessary to ensure the scalability for high-parallelism levels with many complex design tradeoffs that have to be taken into account depending on the actual requirements of the application. For example, the hierarchical data-identification-based partitioning with single/multi-stage switch combination provides almost as high as 5 Gbps of throughput with much smaller area compared to the other two-level partitioned single-/multi-stage switch combinations.

For high-performance applications, a state-of-the-art partially parallel LDPC architecture is proposed by Liu and Shi [36]. The architecture exploits partially parallel microarchitecture for CNP and fully parallel microarchitecture for VNP with the macroarchitecture parallelism of 84 for each kind of processors (CNP and VNP). To make a fair comparison of this architecture to architectures constructed by our method, we assumed a standard set of parameters (IEEE 802.15.3c code, code length (672 bits), code rate (7/8), number of iterations (10), and data width (8 bits)). We then implemented the architecture proposed in [36] and our architectures in the same technology (TSMC 90 nm LPHP standard Cell Library) using our DSE and synthesis tool called multiprocessor accelerator explorer (MPA-explorer). The throughput achieved by the architecture proposed in [36] is limited to 1 Gbps due to the long delays of the flat multiplexer- and demultiplexer-based interconnects scheme exploited in [36]. Our architecture with the same processing parallelism implemented using our MPA-explorer achieves *1.94 times* higher throughput, *1.43 times* lower area, and almost the same power consumption compared to the architecture proposed in [36]. The architecture generated by our tool incorporates a hierarchical partitioned network instead

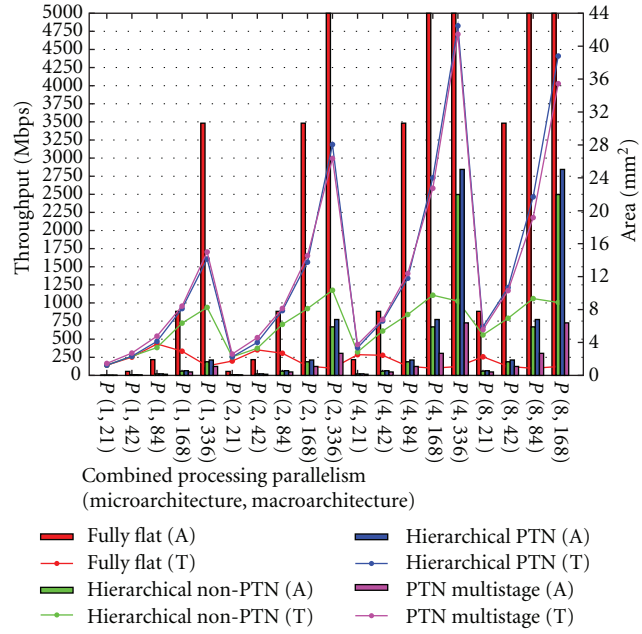

Figure 15: Combined results representing interconnect parameters for various processing parallelism.

of the flat multiplexer, and demultiplexer-based interconnect scheme proposed in [36]. This is the main reason for almost the twice higher throughput by almost 50% lower area of the LDPC decoder architecture generated by our MPA-explorer. Most of the architectures proposed in the literature [22–26] are low-throughput architectures and correspond to the first point, the tuple $P(1,21)$, in the results in Figures 10 and 14.

Summing up, the above-discussed memory and communication architecture design approaches, when applied in combination, ensure scalability of the memory and communication architecture for the massively parallel multi-processor hardware accelerators and exploration of the complex design tradeoffs. This reconfirms once more the fact that without having a quality-driven model-based design approach, such complex design tradeoffs for massively parallel multiprocessors cannot be adequately explored in a reasonable time.

5. Conclusion

In the former sections, we discussed the communication and memory architecture design issues of the massively parallel multi-processor accelerators necessary to realize the required ultrahigh throughput of the highly demanding modern applications. Our discussion was focused on the communication and memory bandwidth and scalability issues. We demonstrated that in the massively parallel hardware multiprocessors, the memory and communication influence on both the throughput and circuit area dominates the processors influence, and the communication and memory design are strictly interrelated. Therefore, communication and memory architecture design is one of the major aspects of our new accelerator design methodology. We

demonstrated that the traditionally used simple flat communication architectures and multi-port memories do not scale well with increase of the accelerator parallelism. In consequence, they are not adequate for the massively parallel accelerators. We proposed to design the application-specific hierarchical partitioned organizations of the communication architectures and vectorized memories exploiting the regularity and hierarchy of the actual information flows of a given application. This drastically improves the scalability. In particular, we demonstrated that for the moderate parallelism levels, the two-level architectures with several small global communication-free clusters or a single global cluster perform well, with performance gains as high as *12 times* and area savings as high as *25 times* compared to the flat communication scheme. However, for the high parallelism levels, only the partitioned hierarchical approach ensures the scalability regarding performance with gains as high as *5 times* and a small area penalty of less than 1% compared to the nonpartitioned two-level hierarchical communication network. To further increase the performance or eliminate the area penalty, the multi-stage and single-stage switch combinations can be employed for local and global switches of the two-level partitioned communication network, resulting in area saving as high as *4 times*. Regarding the memory design, the partitioned vectorized shared (single port) memories seem to be the most promising for the massively parallel hardware multiprocessors. To guarantee the required memory and communication bandwidth and achieve the communication and memory scalability in the whole considered performance range, we incorporated all the discussed in this paper strategies of communication architecture synthesis, as well as of memory partitioning, data distribution, and related data mapping into our multi-processor accelerator design method and related automated architecture exploration framework. Using this framework we performed a large set of experiments including all the experiments referred to in this paper. The experiments demonstrated that our quality-driven model-based accelerator design method, and specifically its memory and communication synthesis techniques discussed in this paper, are adequate for the hardware multi-processor design for modern highly demanding applications.

Acknowledgment

The authors of this paper are indebted to Marc Duranton for his collaboration and support.

References

[1] L. Jóźwiak, N. Nedjah, and M. Figueroa, "Modern development methods and tools for embedded reconfigurable systems: a survey," *Integration, the VLSI Journal*, vol. 43, no. 1, pp. 1–33, 2010.

[2] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[3] Ieee standard for information technology—telecommunications and information exchange between systems—local and metropolitan area networks—specific requirements. part 15.3: Wireless medium access control (mac) and physical layer (phy) specifications for high rate wireless personal area networks (wpans) amendment 2: Millimeter-wave-based alternative physical layer extension. *IEEE Std 802.15.3c-2009* (Amendment to *IEEE Std 802.15.3-2003*), pp. c1–c187, 12, 2009.

[4] G. Lechner, J. Sayir, and M. Rupp, "Efficient dsp implementation of an ldpc decoder," in *Proceedings of IEEE International Conference the Acoustics, Speech, and Signal Processing (ICASSP '04)*, vol. 4, pp. 665–668, 2004.

[5] G. Falcao, L. Sousa, and V. Silva, "Massively LDPC decoding on multicore architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 2, pp. 309–322, 2011.

[6] L. Jóźwiak and Y. Jan, "Quality-driven methodology for demanding accelerator design," in *Proceedings of the 11th International Symposium on Quality Electronic Design (ISQED '10)*, pp. 380–389, March 2010.

[7] R. Schreiber, S. Aditya, B. R. Rau et al., "High-level synthesis of nonprogrammable hardware accelerators," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pp. 113–124, July 2000.

[8] K. Kuchcinski and C. Wolinski, "Global approach to assignment and scheduling of complex behaviors based on HCDG and constraint programming," *Journal of Systems Architecture*, vol. 49, no. 12–15, pp. 489–503, 2003.

[9] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau, "Spark: a high-level synthesis framework for applying parallelizing compiler transformations," in *Proceedings of the 16th International Conference on VLSI Design*, pp. 461–466, 2003.

[10] Z. Guo, B. Buyukkurt, W. Najjar, and K. Vissers, "Optimized generation of data-path from C codes for FPGAs," in *Proceedings of the Design, Automation and Test in Europe (DATE '05)*, pp. 112–117, March 2005.

[11] W. Sun, M. J. Wirthlin, and S. Neuendorffer, "FPGA pipeline synthesis design exploration using module selection and resource sharing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 254–265, 2007.

[12] S. P. Mohanty, N. Ranganathan, E. Kougianos, and P. Patra, *Low-Power High-Level Synthesis for Nanoscale CMOS Circuits*, Springer, 1st edition, 2008.

[13] A. Canis, J. Choi, M. Aldham et al., "LegUp: high-level synthesis for FPGA-based processor/accelerator systems," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '11)*, pp. 33–36, 2011.

[14] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: from prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[15] M. Tudruj and L. Masko, "Communication on the fly for hierarchical systems of chip multiprocessors," in *Proceedings of the 6th International Symposium on Parallel Computing in Electrical Engineering (PARELEC '11)*, pp. 19–24, 2011.

[16] L. Wang, C. Ding, S. Zhong, and J. Zhang, "GNLS: a hybrid on-chip communication architecture for SoC designs," *International Journal of High Performance Systems Architecture*, vol. 3, no. 2-3, pp. 157–166, 2011.

[17] M. Monchiero, G. Palermo, C. Silvano, and O. Villa, "Exploration of distributed shared memory architectures for NoC-based multiprocessors," in *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS '06)*, pp. 144–151, July 2006.

[18] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: built for speed," *IEEE Micro*, vol. 26, no. 3, pp. 10–23, 2006.

[19] A. K. Kodi and A. Louri, "A scalable architecture for distributed shared memory multiprocessors using optical interconnects," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, pp. 143–152, April 2004.

[20] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: a quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–20, 2007.

[21] S. Saponara, L. Fanucci, and E. Petri, "A multi-processor noc-based architecture for real-time image/video enhancement," *Journal of Real-Time Image Processing*. In press.

[22] M. Rovini, G. Gentile, F. Rossi, and L. Fanucci, "A minimum-latency block-serial architecture of a decoder for IEEE 802.11n LDPC codes," in *Proceedings of the IFIP International Conference on Very Large Scale Integration (VLSI-SoC '07)*, pp. 236–241, October 2007.

[23] T. Bhatt, V. Sundaramurthy, V. Stolpman, and D. McCain, "Pipelined block-serial decoder architecture for structured LDPC codes," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '06)*, pp. IV225–IV228, May 2006.

[24] K. Gunnam, G. Choi, W. Wang, and M. Yeary, "Multi-rate layered decoder architecture for block LDPC codes of the IEEE 802.11n wireless standard," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '07)*, pp. 1645–1648, May 2007.

[25] P. Urard, E. Yeo, L. Paumier et al., "A 135Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes," in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC '05)*, vol. 1, pp. 446–609, February 2005.

[26] C. Studer, N. Preyss, C. Roth, and A. Burg, "Configurable high-throughput decoder architecture for quasi-cyclic LDPC codes," in *Proceedings of the 42nd Asilomar Conference on Signals, Systems and Computers (ASILOMAR '08)*, pp. 1137–1142, October 2008.

[27] S. Kim, G. E. Sobelman, and H. Lee, "Flexible LDPC decoder architecture for high-throughput applications," in *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems (APCCAS '08)*, pp. 45–48, December 2008.

[28] Y. Sun, G. Wang, and J. R. Cavallaro, "Multi-layer parallel decoding algorithm and vlsi architecture for quasi-cyclic LDPC codes," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 1776–1779, 2011.

[29] P. Radosavljevic, A. De Baynast, M. Karkooti, and J. R. Cavallaro, "Multi-rate high-throughput LDPC decoder: tradeoff analysis between decoding throughput and area," in *Proceedings of the IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '06)*, pp. 1–5, September 2006.

[30] M. Cocco, J. Dielissen, M. Heijligers, A. Hekstra, and J. Huisken, "A scalable architecture for LDPC decoding," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '04)*, vol. 3, pp. 88–93, 2004.

[31] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[32] L. Jóźwiak, "Quality-driven design in the system-on-a-chip era: why and how?" *Journal of Systems Architecture*, vol. 47, no. 3-4, pp. 201–224, 2001.

[33] H. Zhong, T. Zhang, and E. F. Haratsch, "VLSI design of high-rate quasi-cyclic LDPC codes for magnetic recording channel," in *Proceedings of the IEEE 2006 Custom Integrated Circuits Conference (CICC '06)*, pp. 325–328, September 2006.

[34] X. Y. Shih, C. Z. Zhan, C. H. Lin, and A. Y. Wu, "An 8.29 mm2 52 mW multi-mode LDPC decoder design for mobile WiMAX system in 0.13 μm CMOS process," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 3, Article ID 4456789, pp. 672–683, 2008.

[35] K. Zhang, X. Huang, and Z. Wang, "High-throughput layered decoder implementation for quasi-cyclic LDPC codes," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 6, pp. 985–994, 2009.

[36] L. Liu and C. J. R. Shi, "Sliced message passing: high throughput overlapped decoding of high-rate low-density parity-check codes," *IEEE Transactions on Circuits and Systems I*, vol. 55, no. 11, pp. 3697–3710, 2008.