Survey of Approaches for Successful Design and Programming of Multiprocessor Systems on Chip

Krste Mitric, Miodrag Bolic and Voicu Groza

School of Information Technology and Engineering
University of Ottawa
{kmitr027, mbolic, vgroza}@uottawa.ca

Abstract

Multiprocessor System on Chip (MPSoC) has become an ultimate solution for many applications with high processing requirements over the last couple of years. Multimedia, Network Processing, Gaming and Automotive are just some of examples where solutions using a single processor hit performance and power consumption wall, despite the trend of performance increase with newly emerging processors. Strict requirements for high processing power and low power consumption for some of these applications can only be met using heterogeneous MPSoC, with different types of processors targeting appropriate segments of the application. A correct interconnection topology with proper communication, including data exchange and synchronization, appears to be the key for meeting application requirements. Therefore, the interconnection among processing and memory elements has the largest weight in MPSoC design process. In this paper we discuss MPSoC design with emphasis on processors interconnection as the key for successful mapping of the parallelized application software onto multiple processing elements. We focus on issues, challenges and proposed solutions by leading experts from industry and academia in the area of MPSoC design methodology and automation. In addition to interconnection, we discus the design space exploration, types of selected processors that determine the system architecture, power consumption, parallelizing sequential software and mapping of the software onto MPSoC - issues that system architects and designers face when creating an MPSoC.

1 Introduction

For many newly emerging applications ~~that~~with~~ require more processing~~increasing processing requirements and ~~smaller~~decreasing power consumption needs~~with increasing processing and stricter power consumption requirements~~, MPSoCs appear to be the only viable implementation choice. Knowing that the maximum achievable ~~speed~~ clock rate with current embedded processor is around 1 GHz, and that some applications, like baseband processing, require over 10 Giga Instructions per Second (GIPS) [11], it is obvious that a single-processor system is not a solution for such applications. In order to achieve such speed, multiple application specific processors have to be employed, and ~~with great level of~~ parallelism that ~~must originate in~~is inherent from the ~~chosen~~ application has to be utilized. First dilemma for architects and designers of such a system is whether to choose multiple processors of the same type (homogeneous system) or to employ different types of processors, each suitable for a specific segment of the application (heterogeneous system). From the perspective of application software development and mapping process, reusability and maintenance, homogeneous systems are always desirable. The problem is that in many cases they can not achieve processing and/or power consumption requirements, so heterogeneous systems must be used.

A noticeable trend in the last couple of years is that highest performance single processors are being replaced by multi-core processors, mostly dual-core processors. Despite the fact that they have more than one processor on the chip, these multi-core processors are not considered to be MPSoCs since they are not systems intended to solve a certain problem or a set of related problems. Their hardware and software interface is designed such that they appear as a single processor, so they fit in single-processor systems as well.

In order to get full advantage of the parallel nature of MPSoCs, achieving required processing speed by running maximum of the application threads in parallel, the processors interconnection has to be designed appropriately. Such an interconnection must be capable of supporting exchange of data and control/synchronization messages between the threads, avoiding communication bottleneck and processors idling while waiting for data to be processed. A chosen interconnection type can heavily bound the system's performance and it is one of the key decisions in every MPSoC design. Once the interconnection network type is specified, the choice of network elements becomes critical for the overall network and MPSoC performance. For achieving the best performance by employed processing elements in an MPSoC, communication tasks have to be separated from the computation tasks, offloading main processing elements from communication activity involvement. To achieve such separation, it is usually required from interconnection network elements to have some processing capabilities.

For execution of some 'processing power hungry' applications, MPSoCs are required to have tens or even

批注 [FoM2]: In order to achieve such speed, multiple application specific processors have to be employed with great level of parallelism that must originate in the chosen
Maybe: In order to achieve such speed, multiple application specific processors have to be employed and parallelism that is inherent to the application has to be utilized.

批注 [FoM1]: Rewrite

hundreds of different processing elements, multiple memory modules and an appropriate interconnection network. Processing elements can be general-purpose processors, programmable application-specific coprocessors, non-programmable processing modules, or any combination of the formerly mentioned. With increasing the number of processors in a system, the increase of hardware complexity is most obvious in the interconnection network, especially in the case of heterogeneous systems. However, for a class of applications with strict low-power requirements, a heterogeneous system is the only feasible solution.

In most cases, MPSoC is used as a solution when the requirements cannot be met with a single off-the-shelf processor. Hence, all requirements for performance, power consumption and/or cost could not be satisfied. In order to find the appropriate trade-off of performance vs. power consumption and/or cost, it becomes a must to perform Design Space Exploration (DSE) prior to implementation. One of very important goals in every MPSoC design process is to minimize the number of manual intervention steps by designers, as well as the time before an optimal or satisfactory solution is found.

Mapping of the application software onto such a large parallel system is not trivial. One of biggest challenges appears to be how to reuse existing application software and map it properly to MPSoC. Designers are faced with high complexity problems during parallelization of the sequential application software. The parallel software is then mapped to MPSoC. The parallelization and mapping processes should interact with hardware design, requiring a number of appropriate steps in each phase. Feedback information exchange, in form of ‘ping-pong’, should be considered between high level system architecture vs. software mapping and parallelization. Such complexity in designing both, hardware and software requires proper methodology to be specified and a certain level of automation, to avoid loss of the control of the design process.

All steps toward creating an MPSoC appear to be a big challenge for system architects, software and hardware designers, as well as for test and verification engineers. MPSoCs are usually tailored for specific applications and therefore their complexity is application-dependent. Simple MPSoCs are generally used to just lower the cost and power consumption of the multiple chip solution. In these cases, a traditional master-slave bus can be used to interconnect the processors. The bus becomes a limiting factor for system performance when the number of processors increases. Hence, arbitration becomes too complicated, capacitive load on the bus becomes too high, therefore limiting the bus clock frequency. In addition, competing for memory access by the processors becomes timely intensive, ~~which leads~~leading to decreas~~inge~~ of the overall processors’ performance.

In this paper we summarize recent research efforts that have been focused on identifying a good methodology for MPSoC design, with emphasis on communication among processing elements and advantages/limitation imposed by the interconnection choice on all other aspects of MPSoC design.

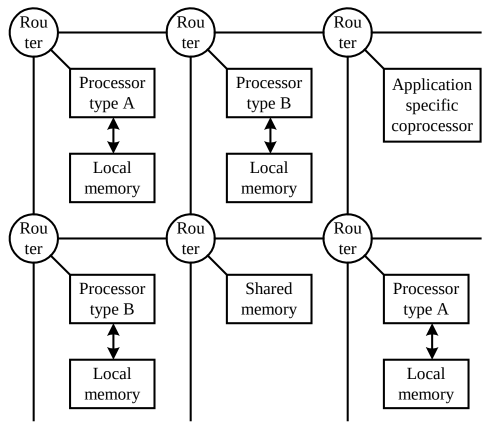
This paper consists of 6 sections. Section 2 addresses processors’ interconnection, as one of major challenges in the entire MPSoC design process. In Section 3 we describe design space exploration challenges and advantages when designing MPSoC. Section 4 deals with parallelizing application software that had been created as sequential. In Section 5 we discuss attempts to perform automation of MPSoC design and with Section 6 we conclude the paper and give suggestions for future work.

2 Processors interconnection

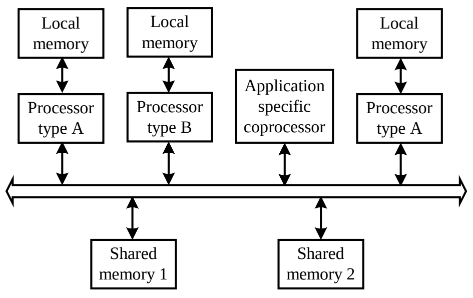
In this section we analyze MPSoC processing elements interconnection, examine the design issues and challenges, and discuss improvements in both hardware and software, suggested by academia and industry researches. ~~Mainly, ~~We identified ~~are ~~seven major areas ~~with contributions for interconnection improvement~~in the research on interconnection networks for MPSoC. Many of the examined researches were addressing more than one area:

a) Building of hardware and/or software communication controller to improve interconnection and overall MPSoC performance by offloading processors from performing communication tasks. This area is primarily addressed in [9], [10], [14] and [17].
b) Modeling to allow proper interconnection design. Different mechanisms and modeling levels are suggested in [15], [19] and [26].
c) Performing application software mapping in parallel with interconnection design to improve utilization of a parallel nature MPSoC, which is addressed in [9], [10] and [19].
d) Introduction of specialized hardware elements that can significantly improve interconnection performance. Larger throughput routers and specialized register files are suggested by [12] and [24], respectively.
e) Network on Chip (NoC) design, latency issues and implementations are addressed in [3], [12], [18], [19], [23] and [26].
f) Design for power, as ~~leading~~important methodology in interconnection design is specifically highlighted in [12], [18] and [23].
g) Deadlock avoidance methodology, caused by interconnection network latency is addressed in [3], [17] and [25].

Interconnection network is the base of every MPSoC. In almost all cases, topology and characteristics of the network determine the system performance. Having in mind that a majority of MPSoCs are designed and tweaked to address specific applications ([1], [3], [4], [6], [8], [10], [11], [13], [15], [17], [19], [20], [22], [24], [26]), the choice of network type, topology, routing methods and mapping of the application software must be made based on the application that is being addressed. The application usually dictates the size of the network, as well as potential locality that can be exploited. In general, traditional on-chip interconnection networks (usually bus-based) can be exploited in MPSoC as long as the number of network elements is small. Whenever possible, bus-based architecture is suggested due to its simplicity, high flexibility and number of available software tools that can be partially or fully reused, depending on targeted application. ~~Network on Chip (~~NoC~~)~~ should be considered as a viable option when the number of network elements is large. It comes with a price of often unpredictable and long latency; hence, additional complexity is imposed to application mapping. NoC designers should always try to make the NoC hardware to be viewed as a homogeneous interface by application developers, in all cases, including heterogeneous MPSoC ([10], [19]). Figure 1 shows two examples of multiprocessor interconnection networks.



(a)



(b)

Figure 1: Heterogeneous system interconnection network examples: (a) 2D mesh (b) bus based

Designing of interconnection network is seen as main challenge by most of MPSoC design researches. In [14], Liang et al. approached the problem of MPSoC design from the software layer starting with the application software that had been written for sequential single processor execution. Connection between processing elements is seen as the system performance bottleneck for most of the high performance MPSoCs~~,~~. ~~and s~~Suggested ~~is~~ was an appropriate topology and communication protocol for resolving the bottleneck by using new ~~intercommunication~~ interconnection architecture.~~, while sticking with heterogeneous MPSoC, scalable up to tens of processing elements.~~ Presented is a stream-based point-to-point communication among processing elements, with messages exchanged between two processors defined as streams. A separate communication controller, with its instruction memory that contains precompiled scheduled stream events, is used to support the communication. Each network clock cycle, different stream can use the communication channel. Every clock cycle, communication scheduler determines which stream will use the physical channel, allowing stream pipelining from source to destination. It is important to note that all schedules (stream definitions) have to be loaded prior runtime, and only the decision of which schedule to execute is made at runtime. This way, communication overlaps with computation since the communication controllers take care of communication, allowing processing elements to perform computations. In [10], Jerraya et al. also insist on separating communication from computation in complex heterogeneous MPSoC, presenting communication-centric approach. Actually, proposed is an additional separation of the communication design from the overall system design. The role of communication designer, sometimes referred to as an architecture designer, is to design on-chip communication network by splitting communication design tasks between hardware network design, design of software communication layers and operating system design. Main role of system designer is to extract enough information from chosen application to be able to clearly separate communication tasks from main data processing tasks, and perform proper tasks separation among hardware, software and communication. The communication consists of two main tasks: wrapper design and network design. Role of the wrapper is to adapt communication protocol of particular component to the main communication network protocol, previously chosen by system and/or network designers. Each wrapper is composed of both, software and hardware. Besides the internal module bus, the hardware part must include a processor adapter and channel adapter. The software part must include drivers to control the wrapper hardware and some high level communication primitives and services, like task scheduling and interrupt management.

For successful interconnection design, modeling of the communication elements is desirable for different design stages. In [26], Wieferink et al. presented system where communication interface can be modeled using three topologies: point-to-point, bus or NoC. Transaction Level Modeling (TLM) for communication interface is recommended in the early stage of the design, then moving to functional cycle accurate level. For the most accurate level, users are offered pin-level, or RTL level of modeling. In [15], Lyonnard et al. presented three different communication model levels. The highest level is system level, which can just exchange messages among processors using simple send and receive commands. The second level, macro-architecture level uses channels for its communication medium, with communication parameters assigned. Channels have their own communication protocols (e.g. FIFO, handshaking, etc.). The third level is micro-architecture level, which is cycle accurate communication model. Communication wrappers are implemented in forms of communication coprocessors, in either hardware or in software or mixing the two.

批注 **[FoM4]:** How could we classify the papers?
Fixed scheduling vs. runtime scheduling
Communication type: streaming, ...
Levels of design: hardware, software communication layer, operating system
Modeling:

Znaci, glavna zamerka je da radovi nekako nisu povezani. Kada bi mogla da se napravi neka tabela u kojoj bi vrste bile radovi a kolone modelining, scheduling, ... onda bi mozda mogli da se uporede radovi. Drugo, nije jasno sta je rezultat tih radova i kako autori radova porede to sto oni rade sa drugima.

批注 **[MSOffice5R4]:** Pokusacu da napravim takvu tabelu sa mogucnosti koriscenja ‘checkmark’-a za radove koji zadovoljavaju ono sto pise u kolini. Jedino se bojim da tabela ne postane prevelika (mnogo kolona) jer ima puno toga sto su razni pokusali da urade, i nema bas toliko toga zajednickog. Ne znam kako cu da stavim takvu tabelu u ovaj papir (na pola sirine strane). Smislicu vec nesto.
Sto se rezultata tice, neki od njih porede rezultate sa svojim prethodnim rjesenjima, neki samo sa nekim drugim autorima, neki cak samo navode sta su bili rezultati 9za koliko vremena je procesuirana oderdjena aplikacija). Svako od njih (mozda samo par izuzetaka) navodi sta su rezultati, ali sam mislio da je to vec preveliko detaljisanje ako pocnem da pisem koje konkretne rezultate je svako od njih postigao, Nekako mi je bilo blize da navedem sta su koristili, koje su ideje imali za poboljsanje necega (brzine, smanjenje potrosnje, …)
Ne znam sad, da lid a se osvrnem bar na neke od rezultata?

批注 **[FoM3]:** This is too long and unclear. Interconnection or intercommunication

Optimization of the interconnection network design is very important in the process of achieving application performance goals, with the emphasis on application mapping methodology for a given network topology. The MPSoC design needs to be performed in parallel with ~~application~~ mapping ~~process~~, producing in the end of the design process an optimal ~~network~~ design ~~with choice of mapping strategy that can achieve the best results~~ for a given application ([3], [19], [20], [26]).

In complex heterogeneous MPSoCs, network interface design can be considered for every processing and memory element. This way, every system component can look as homogeneous from network access and communication perspective. In [9], Grasset et al. suggested that the global network protocol should be defined by the system designers, and interconnection among processing elements and any other element in the system must ~~be established using that communication~~obey the same protocol. So, each component requires network interface design. A composition method that uses component libraries and composition tool is recommended for interconnection network design. Library elements can be selected based on given system architecture description, then assembled and implemented. The main disadvantage of this method is a requirement for very large libraries in order to support many available network elements and topologies. Network interface generation goes through three major steps: functional analysis, architecture generation and implementation. The first step translates network interface specification into abstract network interface that defines basic operations like arbitration, address translation and error detection and handling. The architecture generation is mainly translation of the abstract components into generic components and generation of interconnection topology between them. Generic component is defined as component that has known behavior and ports. This step enables exploration of different architectures. The implementation step assumes mapping generic components into given RTL library elements.

Often identified as the bottleneck of the MPSoC, the interconnection network sometimes can not be improved enough using readily available communication elements, so required are some component changes that mainly increase communication capacity. Efficiency of the processors' communication is always dependent on the chosen network topology. When message passing is used as the method for exchanging data between processors on an MPSoC, it is critical to ensure fast data exchange. Message passing can be done through shared memory, that is addressable by all or a group of processors, but due to relatively slow access time, and a very limited number of access ports (in most cases the maximum is two), the access to memory becomes the main bottleneck. In [24], Tabrizi et al. moved the message passing one layer closer to the processors and incorporated it ~~message passing~~ directly into processors local registers. Processors communication on MPSoC was enhanced by creating a multiple port register file composed of multiple registers - to be shared by local and remote processors.

NoC paradigm became unavoidable in large heterogeneous MPSoC design ([3], [12], [18], [19]). Similarly to traditional multiple chip interconnection networks, packed-based communication is commonly used, with a worm-hole routing approach, suitable for systems on chip that usually contain small memory blocks per network element, enough to store just couple of packet flits.

In [19], Paulin et al. presented a platform that is capable of mapping application software onto heterogeneous platform, providing homogeneous environment. The main element of the presented platform is an orthogonal, scalable NoC that has predictable latency, supporting different network topologies like crossbar, bridges and busses. In order to overcome issues with communication latency and to achieve required execution speed, model of processor multithreading is being used. The multithreading allows the same processor to execute other tasks when the existing thread is being blocked due to on-platform communication latency.

Design for power paradigm has to be exploited in MPSoC design when power consumption is one of the key requirements. Like in other MPSoC design methodologies, interconnection network design can be seen as most critical in achieving power consumption goals of entire system, and therefore the starting point in the design process. Network elements are usually seen as the bottlenecks in processor communication and hence in achieving system processing ~~power~~ goals. Choice of the faster network elements (with higher throughput), in combination with the slower network elements can improve the network performance, but increase power consumption. This way, the communication network itself becomes heterogeneous. If the communication network is carefully designed such that processing elements which communicate more are grouped together, required processing performance can be achieved with low power consumption if mixture of fast and slow network elements is used. The fast elements with higher throughput (with more physical communication channels) have ultimately higher power consumption. Such elements can be employed in communication intensive parts of the network, while the cheap and slow ones can be used elsewhere. In [12], Kreutz et al. examined mesh NoC with proper distribution of processing elements over the network. An appropriate optimizing algorithm is used to select between the given three routing elements. The routers are designed to fit mesh NoC, with one local connection and four connections to the neighboring routers (north, south, east and west). The fastest router has a dedicated physical channel for each connection, while the slowest one has only two physical channels, one being used by local processor and the other one to be multiplexed in time among four connections.

批注 **[FoM6]:** ??? with mapping

批注 **[FoM7]:** optimal network or optimal design. How can design be optimal?

批注 **[FoM8]:** What is the best result? Tradeof between area and processing speed?

批注 **[FoM9]:** ?? must obey to the same protocol??

批注 **[FoM10]:** What is the difference between this and the wrappers

批注 **[MSOffice11R10]:** No difference, authors called it that way.

批注 **[FoM12]:** Why is this composition method

批注 **[MSOffice13R12]:** I would call it synthesis, but authors strictly said that they considered 'synthesis' and composition methods. As the synthesis method they assume a method that starts from top level and big modules are synthesized into small modules, then these to smaller, and so on. They find it not so good and recommend this one, which they call composition. I would hesitate then to call it synthesis and mention their name.

批注 **[FoM16]:** Why? Do only some of the network elements become heterogenious?

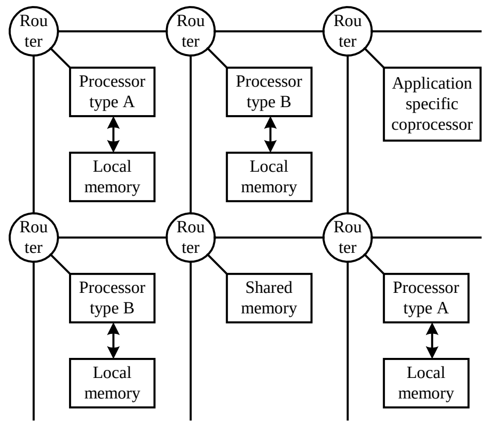
批注 **[MSOffice17R16]:** When you use same elements (e.g. routers in heterogeneous MPSoC, that has different type of processors, then the network itself is homogeneous. But, when the network elements are of different type, then the network itself becomes heterogeneous.

批注 **[FoM18]:** IF we properly map processors to have localcommunication, why do we need fast and slow network elements?

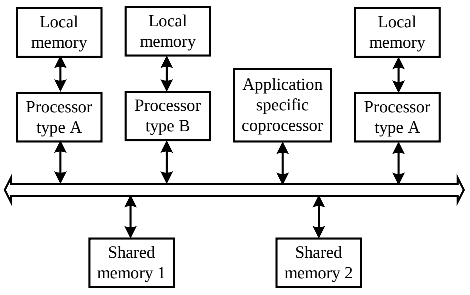
批注 **[MSOffice19R18]:** Fast elements in this case are nothing but the elements with more channels. So, for the area of network with higher data throughput (localized if mapping is properly done), we need network elements with more channels in order to achieve max speed. The area with less traffic can be satisfied with elements that have less channels (and they consume less power).

批注 **[FoM14]:** Did they improve communication speed? By how much?

批注 **[MSOffice15R14]:** This is one of example when results are inconclusive. They first said that to do one MPEG4 application, you need 21 ARM @ 200 MHz, then they map 83% of code in couple of HW specific modules ('keeping flexibility' as they claim). Then they compared execution of MPEG application on 4 ARM processors and their HW components varying number of allowable threads per processor, and they compare it with theoretical performance (number of processed frames per second) that assumes no delay in communication. There is no comparison with another type of network.



(a)



(b)

~~Figure 1: Heterogeneous system interconnection network examples: (a) 2D mesh (b) bus based~~

Deadlock is a phenomenon that can appear in MPSoC that use NoC as a communication network, and it can happen due to different network delays among processing elements and ability that a faster element can 'jam' the network element before the slower element has time to respond in situation when they share the network element, e.g. router. Means of synchronization have to be used in such situations, and the easiest solution is to use physically separate control lines for the network control. A more expensive solution would be to increase network capacity by adding virtual channels. In [3], Bartic et al. examined mesh network and suggested changes necessary to be performed on application (created for single-processor system) because of the network latency, when the application is being mapped to the MPSoC with NoC as the communication network. Examined were also methods for avoiding deadlock that can happen in such NoC. Adopted was a method of using additional data storage buffers for the critical processing elements. These buffers are used for storing extra data from faster processing elements that might appear due to unequal network latency, allowing slower processing elements to deliver their data and avoid the deadlock. Interesting approach for examining deadlock avoidance and improving of multiprocessor interconnection network performance in presence of faulty nodes is examined in [25]. Vaidya et al. presented fault-tolerant routing methodologies that could be applicable to NoCs when some network topologies, like mesh, are employed. With standard routing methodologies, if a node becomes faulty, a large part of the system has to be deployed in the best case scenario, or total system fault is considered in most cases. Fault-tolerant routing allows just insignificant performance degradation in many situations. There are many known algorithms for deadlock avoidance when fault-tolerant routing is used. Many of them use separate virtual channels for request and response messages.

Natural connection between processing elements for data flow dominated applications, like signal processing and multimedia, is connection using FIFO channels because these applications have tasks communicating to each other through streams of data. The easiest way for synchronization among processing elements communicating through such interconnection network is using blocking read and write software primitives. In [17], Nikolov et al. presented MPSoC development platform that can implement three different types of processors interconnection networks: point-to-point, crossbar switch or shared bus. The composition method is used to assemble interconnection network from available library components. Glue logic, called communication controller, is used for synchronization between procedures at the hardware level. In addition to its local program and data main memory, each processing element has its own connection memory, used for data communication and synchronization between processing elements. Each processor writes only to its own connection memory and uses the communication controller to read from other processors' connection memories. FIFO sizes are chosen to guaranty deadlock free communication, but they have to be carefully calculated (minimized) to not to overdesign and increase cost and power consumption of the MPSoC. Such calculation is result of thread scheduling algorithms used to ensure that every data element is written before it is read. Greedy approach was used for the thread scheduling. Blocking read and write primitives are used for synchronization, allowing processor to see FIFO channel as two memory locations in its connection memory address space. The first location is for read/write data and the second one is for empty/full status flag.

The idea of switching network elements on and off depending on their usage becomes challenging for power-aware NoC~~s~~ design.~~, having in mind that s~~Switching time of embedded routers is reasonable small, unlike the classical multiprocessor networks where switching time of stand alone routers is too big that ~~to~~ undermines potential saving achieved when the router is not used. In [23], Soteriou et al. demonstrated that power saving in NoC can be up to 90% when on/off time is in order of 10 to 100 network cycles. Routing protocols have to be carefully chosen to avoid deadlock. To avoid deadlock, the chosen mesh network was split in two virtual networks: west-last and east-last, ensuring that packets managed with one virtual network will never cross to the other. Sleep and wake-up mechanisms had to be created using network statistics to determine the appropriate time for switching off/on with different threshold for each.

批注 **[FoM22]:** Is this shared memory?

批注 **[MSOffice23R22]:** No, each processor has its own. Processors access the other processors' memories through the communication controllers

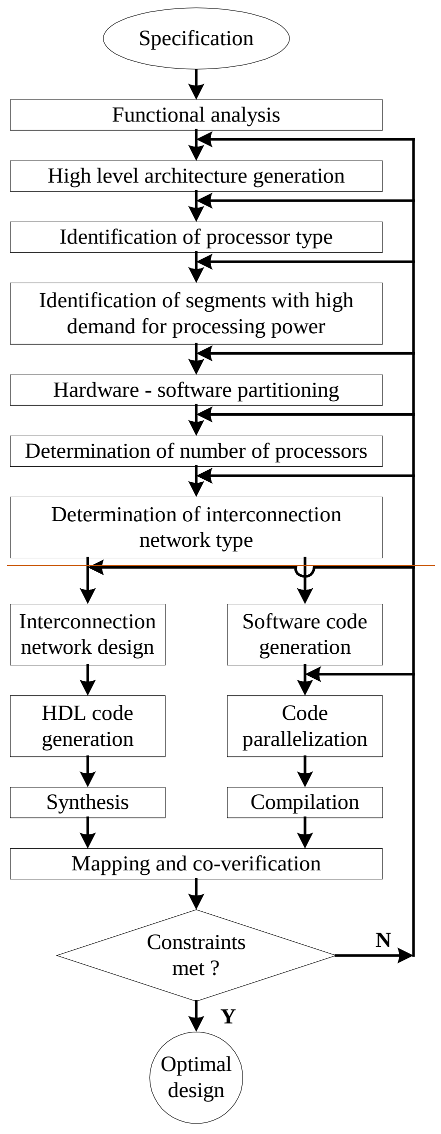
批注 **[FoM20]:** I am confused here with the 3 elements

批注 **[MSOffice21R20]:** This is general (new tab) and has nothing with previously mentioned three type of network elements in that heterogeneous network. I meant to say that problem can occur when for example a processor expects two operands from two different co-processing modules to which it is connected through the same router that has buffer of only 2 memory locations (authors use similar examples if I remember well). If one of them is faster, then the other one will never have chance to send data, and processor will wait for it forever.

批注 **[FoM24]:** Break into two sentences.

An importance of localization of the network elements that have higher communication activity was highlighted in [18], allowing optimal NoC design, using proposed steps in the design methodology.

3 Design space exploration


Specification
Functional analysis
High level architecture generation
Identification of processor type
Identification of segments with high demand for processing power
Hardware - software partitioning
Determination of number of processors
Determination of interconnection network type
Interconnection network design
Software code generation
HDL code generation
Code parallelization
Synthesis
Compilation
Mapping and co-verification
Constraints met ?
N
Y
Optimal design

~~**Figure 2**: Proposed design space exploration flow~~

In this section we highlight importance of design space exploration in an application-driven MPSoC design, and discuss some suggestions presented by industry and academia~~ and recommend our approach for~~. Mainly, the researchers have addressed three areas of the MPSoC design space exploration:

a) Utilization of design space exploration for ~~easy~~ exchange of tasks between hardware and software.
b) Finding balance between performance and power consumption requirements.
c) Importance and methodology for using Field Programmable Gate Arrays (FPGAs) and application specific processors (ASIPs) for MPSoC design space exploration.

Design space exploration appears to be a valuable way for creating a desired design solution for MPSoC based on heuristic methods. Having some of the tools that can automatically generate entire high level system architecture or its parts can be very useful, even if the generated architecture can not closely meet the requirements specified by the application. Such high level architecture can be used to determine some general system performance parameters, which are necessary in an early design stage of the decision-making process. Later, when the system architecture details are specified, the design can be 'tweaked' to meet the application requirements, and even the tools can be fixed accordingly. A good example of such tool generation is a compiler presented in [13]. Leupers et al. suggested that retargetable compilers (compilers that can be applied to another platform requiring ~~a~~ small ~~number of~~ changes) should be considered a solution for future embedded systems. Such compilers can certainly be used in an early stage of system development for design space exploration, since they can be made available in relatively short time. When software and hardware architectures are ready, manual changes to the generated code can be performed in the critical code sections. Then, later the compiler can be changed to perform these manual code changes.

MPSoC design space exploration should be done in a way that should always allow an easy exchange of tasks between software and hardware, as suggested by [4], [7], [15], [17] and [21]. It is very important, especially in the early stages of the design, to determine critical tasks that require high processing power. These tasks usually have to be implemented in hardware. Many 'performance versus power' tradeoffs can be done this way, allowing MPSoC to meet its required performance and cost targets. In [7], Gerin et al. presented an idea of modeling interface between software and hardware using transaction accurate level that can speed up the hardware-software interface simulations. The model can be very useful for exploration of functions and tasks that can be mapped either to hardware or to software, depending what tradeoffs between power and performance need to be made. In [15], Lyonnard et al. suggested automatic generation of low-level components of an application based MPSoC, allowing designers to focus on high-level design, where more can be gained from the application perspective. More design space exploration can be performed, by repositioning some resources inside the design, exchanging functions among design elements and performing tradeoffs between performance and price by assigning some tasks from hardware to software and vice versa.

带格式的: 缩进：左侧： 0 厘米，制表位： 1.5 字符，列表制表位 + 不在 3 字符

带格式的: 项目符号和编号

批注 [FoM25]: When you say solution, what do you mean? Implementation, design, mapping, The ability to simulate at different hierarchy levels with different abstraction layers is the key to successful design space exploration. Different simulation platforms can be used. Most of the used platforms are SystemC or C++ based ([7], [10], [20], [26]). In [26] Wieferink et al. presented CoWare LISA architecture description language for modeling processor architectures, and SystemC for cycle accurate modeling of inter-processor communication and peripherals connected to the bus. LISA environment is capable of automatically generating processor software tools ~~chain~~ (compiler, assembler, linker, instruction set simulator and profiler) that can be used for design space exploration at different layers of modeling abstraction.

~~Field Programmable Gate Arrays (~~FPGAs~~)~~ appears to be very useful for design space exploration during MPSoC prototyping ([5], [14], [16], [19], [22]). Application specific processors are also good tools for design space exploration. In [8], Goodwin et al. presented Tensilica configurable processor capable of generating instruction extension for very large instruction word (VLIW), vector processing and fused operations. Each of these extensions (or all combined) can be used to produce a specific hardware processing module for some critical code functions, performing design space exploration in order to meet processing and power consumption constraints.

Since the power is one of the reasons for MPSoC existence, performing design space exploration with power consumption in mind (or as prime target) is common in almost every MPSoC design methodology ([6], [12], [14], [15], [24]). Some interesting approaches for power saving were presented in [2]. For relocating intensively used parts of code Angiolini et al. suggested ScratchPad memory – simple memory attached to the processor bus that can lower power consumption and higher speed than caches. An interesting point is that power saving, using this technique can be achieved over executable binary code, without requirements for access to either source code or compilers.

A very interesting use of MPSoC for solving math problems that could not be solved using general-purpose processors was presented in [22]. Prasanna et al. presented a good example of how reconfigurable logic can be used in MPSoCs, by applying design space exploration technique, to generate application specific coprocessor as support for on-chip general-purpose processors.

Proposed design space exploration flow is presented in Figure 2. Iteration steps that can include high level architecture determination, hardware – software partitioning, design of hardware and software as well as simulation or prototyping are performed until given application constraints are met. Depending on the severity of the constraints that are not met (almost met, partially met, badly missed), the next iteration can start from a different point, as presented in Figure 2. Co-verification can be done using hardware-software co-simulation when simulation modeling environment is available, or by data collection from prototyping platform when programmable logic (FPGA) platform is available.

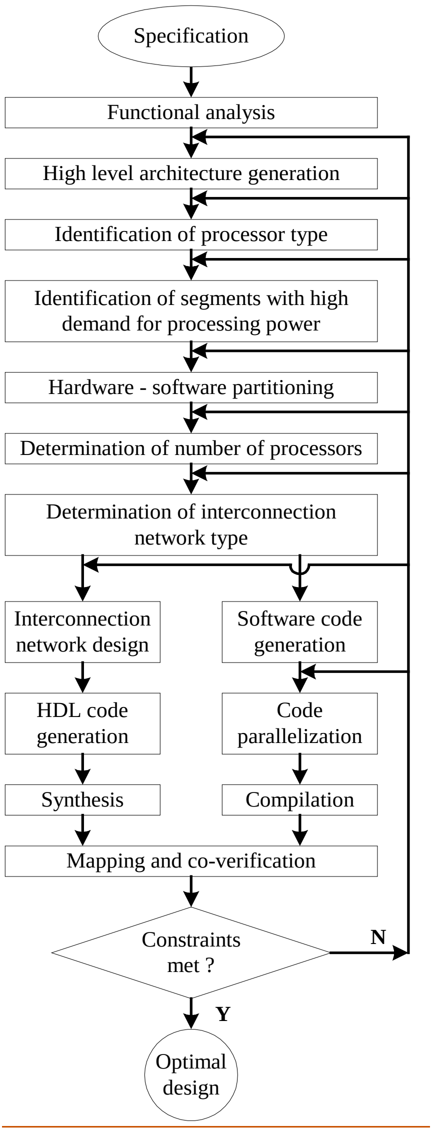

Specification
Functional analysis
High level architecture generation
Identification of processor type
Identification of segments with high demand for processing power
Hardware - software partitioning
Determination of number of processors
Determination of interconnection network type
Interconnection network design
Software code generation
HDL code generation
Code parallelization
Synthesis
Compilation
Mapping and co-verification
Constraints met ?
N
Y
Optimal design

Figure 2: Proposed design space exploration flow

4 Parallelizing sequential software

In this section we address issues and methodologies for proper sequential software parallelization and mapping onto an MPSoC. Mainly, examined are suggested steps by some researchers, that are required for proper code parallelization and mapping, creation of process-based interconnection networks for specific applications that should allow easier parallelization and mapping, and dynamic mapping using runtime tasks allocation.

批注 **[FoM26]:** Put figures below the paragraph is which you describe them.

批注 **[MSOffice27R26]:** Problem was size of figure and available space on page. I will try if for final version.

批注 **[FoM28]:** Bilo bi dobro kada bi se ova slika objasnila na pocetku ove sekcije. Onda bi se za svaki rad koji se opisuje moglo reci na koji segment sa slike se on fokusira i kako obradjeni drugi segmenti.

批注 **[MSOffice29R28]:** Moze, nego ne znam da ne predjem velicimu rada. Oni rekose jos dvije stranice, I nije mi jasno I dalje da li je to 'must' ili moze I vise. Slazem se da bi bilo bolje, mada sam siguran da bi moralo jos ponesto da se doda uz objasnjenje svakog rada I to bi povecalo kolicinu teksta. Doduse, mozda neke svrari mogu da seizbace. Koji bi po tebi bili najbolji kandidati za izbacivanje/skracenje?

Translating to MPSoC parallel platform of already created sequential application software, which was built with a single processor system in mind, appears to be one of the most challenging and time consuming tasks ([11]). Most of the applications have been created by signal processing and communication engineers, who do not usually have good software background, so the application software is highly sequential, without much of a possibility for parallelism. Many researches just overlook the parallelization problem, stating that designers have to manually perform the parallelization task ([3], [4], [10], [19]).

Some attempts are presented to make this process standardized and some techniques are available, such as using graph theory to sort the application software and export some parameters that can be used later for code partitioning, synchronization and scheduling. In [14], Liang et al. prescribe steps to be performed during mapping process. Conversion of sequential application software is followed by code partitioning into basic blocks and their assignment to certain type of processing elements. Then synchronization between processing elements and communication scheduling for determination of communication streams among processing elements, are followed by compilation of basic blocks for assigned processing elements and parallel code generation.

Applications that can be expressed as a set of processes that communicate to each other are examined in [6]. Dwivedi et al. suggested creation of process communication network. Once the processes can be identified, straightforward mapping of sequential to parallel software should be easily achievable, as long as the network was properly designed. Some methodology in the process of parallelizing sequential application software that includes determination of granularity of the application tasks was prescribed in [20].

In [1], Acuaviva et al. addressed issues and challenges with dynamic mapping strategies using run-time task allocation. One of the biggest issues with dynamic mapping is migration overhead, when moving processes between processing elements. Solution was presented by introducing middleware layer that is responsible for implementing task migration in MPSoC, since operating systems are natively designed to run in a single-processor environment. Two strategies are presented: task-recreation, which kills the process on the original processor and recreates it on the new processor, and task-replication, requiring that a replica of each task is present in every processor's local operating system, having the task active only on one processor.

There are lots of research activities with application software mapping to MPSoC hardware ([11]). A compelling methodology was suggested by [3]. Bartic et al. prescribed that the first step is to identify the level of parallelism of the targeted application: either at high level (macro blocks), or at low level (micro instructions). Therefore, suggested are methods for application mapping to MPSoC, based on chosen interconnection network, through modifying the application software to include latency of the chosen network. Another interesting methodology was suggested in [20]. Pazos et al. suggested that the initial step is to perform proper translation of single-processor multimedia application software into smaller parallel models (in SystemC) that can be used for design space exploration of MPSoC. In the next step SystemC library elements are replaced by C++ structures for tasks that will remain as software tasks (that need to be mapped). Hardware modules can be derived from SystemC simulation models, therefore the entire system can be represented using one common simulation model. A different approach, with top-down flow that targets application specific architectures with large amount of data processing, was chosen in [4]. Beltrame et al. prescribed the following methodology steps: static performance analysis, parallelization (to be done manually), validation, simulation and hardware-software partitioning.

5 Automation of MPSoC design

In this section we mainly address reasons for investigating in automation of MPSoC design process, discuss some suggestions by industry and academia in this area, and recommend our solution for the problem. Since many researchers attempted to automate the design process or its part, and just a handful of them presented challenging results, these researches were analyzed in details, emphasizing good ideas, as well as the deficiency of their automation process.

It is very noticeable that almost any step in MPSoC design is application specific ([11]), and many high level decisions, starting from number of PEs, network topology and software-hardware partitioning, have to be done based on the chosen application. Despite the desire for reusability, that always brings advantages for cost and time-to-market, it should be considered as a definition that MPSoCs are application specific. They are in almost all cases required to be used when general-purpose processors systems could not do a good enough job, either from the perspective of power consumption, cost or performance. Except a small set of similar applications that could potentially share the generated MPSoC, there is no room for globalization and generalization of MPSoCs, to make them highly retargetable. Lots of research has been done in the area of MPSoC retargetability, but not so much was produced. One of the great examples of such failure was presented in [5]. Bobda et al. presented solution for "real-life" application. The ~~"real-life"~~ application chosen in the experiment ~~could be anything, since it~~ was forcefully reconfigured in the end without ~~any sign~~ specifying reasons why ~~was~~ the reconfiguring was done~~, except to play with it~~. It is not clear how ~~it~~ the application software was reconfigured, and if that reconfiguration strategy was calculated in run-time. Bit-streams were reloaded into two devices ~~module~~, to show that performance was not affected. Having in mind that application was simple traffic light controller with only two inputs from pedestrian's push buttons, and that the system used 9 processors on an FPGA, measuring performance degradation should not be considered as valid test methodology.

批注 **[FoM30]:** Krle, ovo je suvise ostro i treba ga napisati drugacije.

批注 **[MSOffice31R30]:** Slazem se – pretjerao sam (mislim pretjerao sam da napisem tako nesto, ali stvarno je tako)

批注 **[FoM32]:** who?

There have been many papers that intended to generate tools for automatic MPSoC generation (or at least its network). Almost all of them failed ([6], [15], [16], [19]). Each of them required enormous manual intervention by designers, where simply the word automatic does not fit.

There was a couple of appealing attempts. In [9], Grasset et al. presented bottom-up synthesis method for automatic generation of interconnection interface, based on network interface component library. Great idea and methodology, but even though network interface is considered as one of critical parts of an MPSoC, it is only one element. There are big hardware and software components that need proper automation when designing MPSoC. Another approach was presented in [8]. Goodwin et al. suggested generation of application specific instruction-set processor by automatically generating new data type, new register file to carry such data type and new operations to operate with the new data type. The problem with this proposal is that it creates a heavy burden on hardware by introducing all these extra elements, and also lack of ability to allow designer to influence automation process. In [17], Nikolov et al. presented a platform for design, programming and implementation of an MPSoC. The suggested platform, consisting of a set of parameterized components, is intended for moving design specification and programming to a high system abstraction level. Bottom-up composition approach was adopted. The platform targets data flow intensive applications that can be modeled as Kahn process networks. Despite the great ideas and methodologies, the presented platform has many limitations that reduce its usability. Some of them are presented next. The platform is suitable only for programmable processors as processing elements that can communicate data only through distributed memory. The platform also requires that sequential application code must be written as static affine nested loop program. Platform description of MPSoC topology is required to be created manually, therefore there is no automatic influence on topology ~~by~~ for chosen application. Mapping specification is also required to be manually generated. Communication components used for interconnection network creation are mutually exclusive, so no heterogeneous networks are allowed. Also, processing elements are limited to be of one type, so overall MPSoC has to be homogeneous. Blocking mechanisms are the only available methods for synchronization of processing elements; therefore processor is blocked until communication is granted to it. This can cause performance degradation. Greedy approach that is used for FIFO size calculation might not work for process networks in general, yielding non-optimal buffer sizes, hence higher power consumption and cost. Selection of MPSoC candidates in design space exploration process is also expected to be manual.

6 Conclusion and future work

Designing MPSoC appears to be application driven and application specific. Therefore, methodology for a proper MPSoC design, allowing optimal tradeoff between performance on one side and power consumption and cost (mainly silicon area) on the other, is application dependent. There are ~~two~~ several main drivers behind MPSoC paradigm: inability to perform required application tasks either with general purpose or application specific single processor (due to lack of processing power) and too high power consumption and/or unacceptable cost of the multiprocessor multichip solutions. There are multiple possibilities of how to design appropriate MPSoC. Processing elements can ~~that can be selected~~ include ~~multiple identical~~ general-purpose processors, application-specific processors ~~in combination with general-purpose processors~~, customizable RISC processors, reconfigurable logic, and any combination of these options. A large dilemma in the MPSoC world appears still to be: homogeneous vs. heterogeneous MPSoC ([11], [12]). Generally, it seems that heterogeneous systems can easier meet power consumption and performance requirements, but are harder to deal with, especially from point of software support. Even though there are some forecasts that in a long term, homogeneous systems might prevail, we believe that application-specific problems will always require heterogeneous customized architecture in order to achieve the best performance vs. power consumption and cost.

Based on the chosen interconnection network, the complexity and type (heterogeneous/homogeneous) of MPSoC is determined. A special effort has to be made during high level architecture definition to provision for some kind of simulation or emulation of the interconnection network in order to perform proper design space exploration. Such provision would allow right hardware-software partitioning, which is the key to achieving optimal performance and power/cost. On the other side, designers should make an effort to create simulation platforms to just an essential degree of accuracy. These simulation platforms are desirable ~~P~~primarily in the beginning of the high level architecture stage. At that stage~~, when~~ it is important to have just an accurate enough simulation model ~~that can~~capable of selecting the tasks that have high processing requirements. ~~determine main processing power hungry tasks,~~Such task selection can allow~~ing~~ an initial hardware-software partitioning. After a couple of design space exploration iterations, an effort can be made to improve accuracy of the simulation platform, in order to confirm whether particular application requirements can be met.

In the area of parallelization of previously created sequential application software, generally, not a lot of useful methodologies were proposed. This area is mostly application dependent. We suggest that it would be beneficial to accept application-domain specific approach in which some common parameters of similar application would be extracted, and further parallelization steps would be defined based on the extracted information.

Mapping of application software onto MPSoC appears as an area where some attempts to define good methodology were made, but clearly not enough. Once proper architecture is selected (either before, or concurrently with the mapping stage), a connection between application software and high level architecture should exist, and therefore some kind of properly defined steps could be identified.

批注 **[FoM36]:** This is repeated.

批注 **[MSOffice37R36]:** I agree. But I tried in this section to summarize things, so I should find way to mention these important things … Does this now sound better?

批注 **[FoM33]:** As oposed to what???

批注 **[FoM38]:** Rewrite

批注 **[FoM34]:** Is the message here that there is no automated tools or approaches? We should modify the title of the section.

批注 **[MSOffice35R34]:** There are many attempts, some of them pretty good. Our goal was to show that none of the solutions was good enough for entire design process. The closest was the last explained, and the last sentence is still one of things that are found to be problematic with that approach. I still believe that the title is OK. Or, maybe we can change it to: Attempts for MPSoC Design Automation.
Did you mean something else?

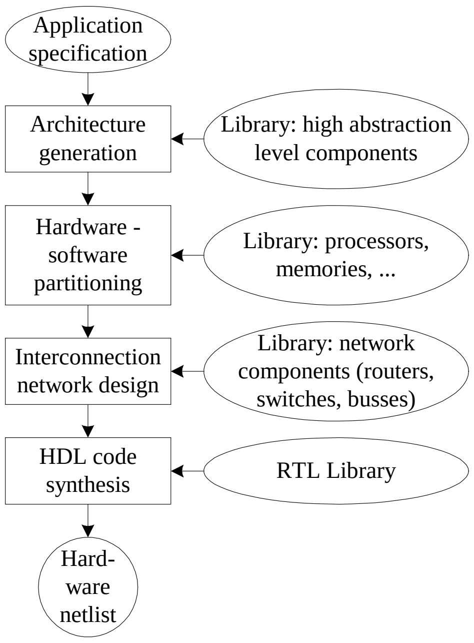

Application specification
Architecture generation
Library: high abstraction level components
Hardware - software partitioning
Library: processors, memories, ...
Interconnection network design
Library: network components (routers, switches, busses)
HDL code synthesis
RTL Library
Hard-ware netlist

Figure 3: Suggested usage of different hardware libraries in process of MPSoC design automation

Automation was the area where many researches claimed to make good progress, but clearly there was no solution offered that could easily be named automatic due to enormous manual intervention required. In some cases researchers simply tried to automate things that could not be combined together due to incompatible nature of the applications. We believe that MPSoC design process automation is the area where progress can be made, using synthesis idea, not only for interconnection network interface for each network module, but for all MPSoC elements, and without many restrictions that prevent some of available platforms from being useful. Figure 3 represents our idea of MPSoC design automation based on synthesis method that uses multiple libraries, each with a different level of abstraction. Applications that have commonalities can be identified, and appropriate libraries can be built. In such synthesis approach, we would like to point out that building proper libraries with many possibilities for type of processor and interconnection network elements available, is the key to proper MPSoC design. Such libraries would be accompanied with appropriate simulation tools, allowing proper design space exploration at different levels of abstraction.

References

[1] A. Acquaviva, A. Alimonda, S. Carta and M. Pittau, "Assessing Task Migration Impact on Embedded Soft Real-Time Streaming Multimedia Applications," *EURASIP Journal on Embedded Sysytems*, Vol. 2008, Article ID 518904, 15 pages, October 2007.

[2] F. Angiolini, F. Menichelli, A. Ferrero, L. Benini and M. Olivieri, "A post-compiler approach to scratchpad mapping of code," *in Proc. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, 2004, pp. 259-267.

[3] T. A. Bartic, D. Desmet, J.-Y. Mignolet, J. Miller and F. Robert, "Mapping concurrent applications on network-on-chip platforms," *in Proc. IEEE Workshop Signal Processing Systems Design and Implementation*, 2005, pp. 154-159.

[4] G. Beltrame, D. Sciuto, C. Silvano, P. Paulin and E. Bensoudane, "An Application Mapping Methodology and Case Study for Multi-Processor On-Chip Architectures," *in Proc. 14th IFIP International Conference on Very Large Scale Integration and System-on-Chip*, 2006, pp. 146-151.

[5] C. Bobda and A. Ahmadinia, "Dynamic interconnection of reconfigurable modules on reconfigurable devices", *IEEE Design & Test of Computers*, Vol. 22, Issue 5, pp. 443-451, Sept.-Oct. 2005.

[6] B. K. Dwivedi, A. Kumar and M. Balakrishnan, "Automatic synthesis of system on chip multiprocessor architectures for process networks," *in Proc. International Conference on Hardware/Software Codesign and System Synthesis*, 2004, pp. 60–65.

[7] P. Gerin, H. Shen, A. Chureau, A. Bouchhima, A. Jerraya, "Flexible and Executable Hardware/Software Interface Modeling for Multiprocessor SoC Design Using SystemC," *in Proc. Design Automation Conference*, 2007, pp. 390–395.

[8] D. Goodwin and D. Petkov, "Automatic generation of application specific processors," *in Proc. International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2007, pp.137-147.

[9] A. Grasset, F. Rousseau and A. A. Jerraya, "Network interface generation for MPSOC: from communication service requirements to RTL implementation," *in Proc. 15th IEEE International Workshop on Rapid System Prototyping*, 2004, pp. 66- 69.

[10] A. A. Jerraya, A. Baghdadi, W. Cesa´rio, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot and S. Yoo, "Application-specific multiprocessor Systems-on-Chip," *Microelectronics Journal*, Vol. 33, Issue 11, pp. 891-898, Nov. 2002.

[11] A. A. Jerraya, O. Franza, M. Levy, M. Nakaya, P. Paulin, U. Ramacher, D. Talla and W. Wolf, "Roundtable: Envisioning the Future for Multiprocessor SoC", *IEEE Design & Test of Computers*, Vol. 24, Issue 2, pp. 174-183, Feb. 2007.

[12] M. Kreutz, A. Cesar, M. Luigi, C. Flavio, W. Altamiro and A. Susin, "Design space exploration comparing homogeneous and heterogeneous network-on-chip architectures", *in Proc. 18th annual Symposium on Integrated Circuits and System Design*, 2005, pp. 190–195.

[13] R. Leupers, "Compiler Design Issues for Embedded Processors," *in Proc. IEEE Design & Test, 2002*, vol.19, no. 4, pp. 51-58.

[14] J. Liang, A. Laffely, S. Srinivasan and R. Tessier, "An architecture and compiler for scalable on-chip communication," *IEEE Transactions on VLSI Systems*, Vol. 12, Issue 7, pp. 711-726, July 2004.

[15] D. Lyonnard, S. Yoo, A. Baghdadi and A. A. Jerraya, "Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip," *in Proc. 38th Conference on Design Automation*, 2001, pp. 518-523.

[16] R. B. Mouhoub and O. Hammami, "Multiprocessor on chip: beating the simulation wall through multiobjective design space exploration with direct execution", *in Proc. 20th*

International Parallel and Distributed Processing Symposium, 2006, pp. 8-15.

[17] H. Nikolov, T. Stefanov and E. Deprettere, "Systematic and Automated Multiprocessor System Design, Programming and Implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 27, Issue 3, pp. 542-555, March 2008.

[18] P. P. Pande, G. D. Micheli, C. Grecu, A. Ivanov and R. Saleh, "Design, synthesis, and test of networks on chips," *IEEE Design & Test of Computers*, Vol. 22, No. 5, pp. 404-413, Sep./Oct. 2005.

[19] P. G. Paulin, C. Pilkington, M. Langevin, E. Bensoudane, D. Lyonnard, O. Benny, B. Lavigueur, D. Lo, G. Beltrame, V. Gagné and G. Nicolescu, "Parallel programming models for a multiprocessor SoC platform applied to networking and multimedia," *IEEE Transactions on VLSI Systems*, Vol. 14, Issue 7, pp. 667-680, July 2006.

[20] N. Pazos, P. Ienne, Y. Leblebici and A. Maxiaguine, "Parallel Modelling Paradigm in Multimedia Applications: Mapping and Scheduling onto a Multi-Processor System-on-Chip", *Int. Global Signal Processing Conference*, 2004.

[21] K. Popovici, X. Guerin, L. Brisolara and A. Jerraya, "Mixed Hardware Software Multilevel Modeling and Simulation for Multithreaded Heterogeneous MPSoC", *in Proc. VLSI Design, Automation and Test*, 2007, pp. 1-4.

[22] V. K. Prasanna and G. R. Morris, "Sparse Matrix Computations on Reconfigurable Hardware," *IEEE Computer*, Vol. 40, Issue 3, pp. 58-64, March 2007.

[23] V. Soteriou and L. S. Peh, "Exploring the Design Space of Self-Regulating Power-Aware On/Off Interconnection Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 18, Issue 3, pp. 393-408, March 2007.

[24] N. Tabrizi and N. Bagherzadeh, "A Distributed and Shared Register File for a Multiprocessor-on-Chip to Support Real-Time Applications," *in Proc. 6th International Workshop on System-on-Chip for Real-Time Applications*, 2006, pp. 215-220.

[25] A. Vaidya, C. Das and A. Sivasubramaniam, "A Testbed for Evaluation of Fault-Tolerant Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, Issue 10, pp. 1052-1066, October 1999.

[26] A. Wieferink, M. Doerper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, G. Braun and A. Nohl, "System level processor/communication co-exploration methodology for multiprocessor system-on-chip platforms," *in Proc. IEE Computers and Digital Techniques*, 2005, Vol. 152, Issue 1, pp. 3-11.