# Parallel Modelling Paradigm in Multimedia Applications: Mapping and Scheduling onto a Multi-Processor System-on-Chip Platform

Nuria Pazos, Paolo Ienne and Yusuf Leblebici
Swiss Federal Institute of Technology Lausanne
Processor Architecture and Microelectronic Systems Laboratories
IN-F and EL-D Ecublens, 1015 Lausanne, Switzerland
Email:nuria.pazos/paolo.ienne/yusuf.leblebici@epfl.ch

Alexander Maxiaguine
Swiss Federal Institute of Technology Zurich
Computer Engineering and Networks Laboratory
Gloriastrasse 35, 8092 Zurich, Switzerland
Email:maxiagui@tik.ee.ethz.ch

## Abstract

*Multi-processor systems have appeared as a promising alternative to face the difficulties of creating even faster uni-processor systems using latest technologies. Emerging design paradigms such as Multiprocessor System-on-a-Chip (MpSoC) offer high levels of performance and flexibility and at the same time promise low-cost, reliable and power-efficient implementations. However, the design complexity of such systems have increased tremendously. One source of the complexity stems from highly parallel heterogeneous nature of the underlying hardware architecture, which poses many challenges for mapping of an application to the architecture. This motivates the development of a unified programming paradigm that facilitates the mapping by hiding the architectural complexity and exposing the parallel resources of the architecture. To enable design reuse, such a programming paradigm has to support a smooth translation of sequentially-coded software algorithms into their parallel implementations. In this paper we address the parallelization of sequential multimedia applications written in C/C++ for their mapping and scheduling onto a flexible MpSoC platform. We show that using our approach an architecture-independent multi-threaded model of a MPEG–2 video decoder algorithm can be obtained with only few modifications to an existing sequential implementation of the algorithm.*

## 1. Introduction

Technological advances have made multiprocessor implementations of embedded systems a viable alternative to traditional uni-processor and pure-hardware designs. Such multiprocessor designs offer high levels of performance and flexibility and at the same time promise low-cost and power-efficient implementations. Nowadays, one of the most promising approaches to design of such systems is a so-called Multiprocessor System-on-a-Chip (MpSoC) paradigm. A canonical view of a MpSoC system consists of a number of processing elements (PEs), which can be programmable processors or fixed application-specific co-processors, and storage elements (SEs) connected to PEs via an on-chip communication architecture. As a result, MpSoC architectures represent heterogeneous systems that offer flexible parallel processing resources for implementation of bandwidth-demanding multimedia applications.

However, new capabilities of MpSoC platforms introduce several design challenges associated with their parallel heterogeneous architecture. A mapping of an application to the architecture starts from a complex system specification, goes through the vast design space exploration and ends with a challenging implementation. In this context the reuse of large base of existing software to perform the exploration of different possible implementations constitutes an important concern. The existing software commonly written in C/C++ language with a uni-processor architecture in mind cannot be directly reused in a *multiprocessor environment* especially in that consisting of a heterogeneous mix of different software and hardware components. The existing software needs to be adapted to the parallel capabilities of the architecture. Furthermore, to enable fast and flexible exploration of the possible application-to-architecture mappings the software cannot be parallelized in an ad-hoc manner, because generation of each new mapping may involve a huge coding effort. Therefore, there is a need for a disciplined approach based on a *unified parallel modelling paradigm* that would enable a smooth translation of existing sequentially-coded software algorithms into their parallel models suitable for the design space exploration of MpSoC platforms.

This paper addresses the problem of creation of such a parallel modelling paradigm. We present a framework and a set of guidelines for transforming existing uni-processor

software for multimedia applications into parallel models that can be then used in the design space exploration cycle of MpSoC platforms. The framework allows to build an architecture-independent multi-threaded model of the selected application, which then can be easily mapped to any homogeneous or heterogeneous multi-processor target architecture. Rather than developing a new domain-specific modelling language, we propose to use SystemC, an existing system-level modelling language. SystemC characteristics, such as its support for different models of computation, its capabilities regarding multi-threading and transaction-level modelling and the fact that it is based on the C/C++ language, make it very suitable for establishing a framework for transformation of the existing sequential software coded in C/C++ into the parallel models for system-level design of MpSoC platforms.

The rest of the paper is organized as follows. Section 2 presents a survey of related work. In Section 3 the proposed parallel modelling paradigm is introduced. Its two main parts, the architecture-independent multi-threaded model and the mapping and scheduling of the application model onto a MpSoC platform, are further explained in detail in Section 4 and 5, respectively. Later on, Section 6 applies the presented modelling method in a case study of a MPEG-2 video decoder multimedia application. Finally, Section 7 concludes the paper.

## 2. State of the Art

The architectural changes introduced in the emerging MpSoCs have a directly consequence in how software engineers programm. This fact has already been acknowledged by several researchers, who have proposed preliminary solutions. Most of them agree on the importance of new high-level programmer views of SoC. In [10], the author makes the analogy between a programmers' view to a heterogeneous multi-processor SoC and an instruction set architecture to a single processing element.

A number of programming models focused on multi-processor SoCs have been presented, such as the MESCAL approach [6], which has served as base for further different programming models. Nevertheless, most of them are application or domain specific, as the one proposed in [7], which only addresses communication modelling. Our approach, instead, is not attached to any specific area of application.

A more general approach composed of two SoC parallel programming models has been introduced in [4]. The *Distributed System Object Component* (DSOC) model and the *Symmetric Multi-Processing* (SMP) model are inspired by leading-edge approaches for large system development, but adapted and constrained for the SoC domain. The authors believe that programming model development will be evolutionary, rather than revolutionary (i.e, rather than the development of entirely new programming paradigms, established software languages and technologies will be supported). The efficacy of these two SoC parallel programming models has only been proven in the area of networking applications yet. Furthermore, they heavily rely on specific hardware for efficient implementation of communication between objects. Rather than that, we do not rely on a self-defined programming model, but we apply an established programming model based on the system-level language SystemC to build the proposed programming paradigm.

Finally, it is also worth to cite the so-called Double-Y methodology developed by IMEC [1], which facilitates the implementation of multi-functional devices supporting a complete application domain. On top of the standard Y-chart approach, they propose an inverted-Y beginning with the definition of the system functionality. This is split in two branches. The first one deals with the design of a flexible architecture, whereas the second branch is supposed to optimise the application code so that it can run on the previously defined architecture (i.e., it transforms a high-level application description into a cleaned multi-threaded description prepared for implementation on embedded systems). But, to date, there is no evidences of applications of such methodology.

## 3. Parallel Modelling Paradigm for MpSoC

Many emerging applications for embedded systems exhibit a high degree of parallelism in their processing. Example of such systems are embedded devices connected to wireless or fixed network and performing some processing of multimedia flows, e.g. with audio and video content. Such systems involve network packet processing functions (e.g. routing, firewalling and encryption) as well as multimedia functions (e.g. compression and decompression of audio and video streams). Both types of functions lend themselves well for a parallel implementation.

In the network packet processing domain, streams of packets only have dependencies among packets of the same flow, but none across different flows. This ensures that the processing of different flows can easily be distributed over several processors. That is, there is an inherent parallelism associated with the processing of separate independent packet flows. Therefore, the performance of the packet processing functions can be increased considerably by splitting large sequential tasks into several smaller tasks and executing them concurrently on several processing units of the architecture.

In the domain of multimedia processing, the parallelism can be found on several levels. Many approaches have exploited parallelism at fine levels of granularity – at the instruction level (in Very Long Instruction Word (VLIW) ar-

chitectures) and at data level (in Single Instruction Multiple Data (SIMD) architectures). However, exploitation of coarse-grain Thread Level Parallelism (TLP) inherent to many multimedia algorithms has not received adequate attention in software-based solutions. This can be explained by the fact that to date single-processor solutions for multimedia embedded systems were prevalent in the development community, and the software was written with a uniprocessor model in mind. However, to fully utilize coarse-grain computational resources of a MpSoC platform TLP in the multimedia applications needs to be identified and effectively exploited as well as the other types of parallelism.

There are some domain-specific languages (e.g. [12]) that support parallel programming concepts, however, most embedded system programmers are not comfortable with them. The programmers are more familiar with C/C++ language and real-time operating systems that provide primitives to build multitasking software. Thus, it is beneficial that the new parallel programming environment is based on the same programming language and programming models as those widely used in the embedded systems' development community. In addition, there is a great potential to reuse a large base of existing sequential software written in C/C++ language.

A further advantage of our approach is that on initial design phases, when it is not clear yet which tasks will be implemented in software and which in hardware, using a single language it allows to build a full system model (or a system specification). Later on, during the design exploration phase, some tasks will migrate into hardware while others will remain in a software implementation. Thus, for those tasks that are remained in software no (or almost no) changes to the code will be required.

The procedure followed during the present work can be divided into two phases. The first stage deals with the description of an architecture-independent multi-threaded model of the original reference software. At this point, a study of the parallelism inherent to the reference software is required. The multi-threaded system specification is described using SystemC language [11] at the selected level of granularity. Later on, the generation of the embedded software for the target multi-processor platform is derived from the previous SystemC multi-threaded model by redefining and overloading the SystemC class library construction elements by typical operating systems functions and C++ supporting structures [5] (e.g. replacing them by POSIX-thread 'pthread' [8] library functions). After this, the second phase comprises the exploration of suitable NoC communication architectures for the MpSoC platform and the mapping of the pthread model onto the target platform. The assessment and evaluation of the results is performed using available benchmarks.

# 4. Architecture-independent Multi-threaded Model

The optimal SoC designs will most likely lie between that of a central controller residing on a single PE and pure data-flow designs with no central controller to dynamically direct resource cooperation [10]. Following this idea, the architecture-independent multi-threaded model proposed in the current work introduces a control flow coordinated across multiple PEs, where multiple data paths are set up by a scope of control flow.

## 4.1. SystemC Modelling

As already introduced before, the characteristics introduced by the system-level language SystemC make it a good candidate for the modelling and simulation of the architecture-independent multi-threaded model. The goal is to take a C/C++ program and transform it into a parallel SystemC executable model. To reach this goal, the following guidelines have to be followed during this phase:

- A decision concerning the granularity of the tasks, into which the system functionality is partitioned, has to be taken. A compromise between coarse and fine granularity has to be reached and be used further on in the modelling.
- Each parallel task is modelled as a SystemC module (*SC_MODULE*), which includes an unique thread process (*SC_THREAD*). The different methods and functions called by a thread are processed sequentially.
- The communication between parallel tasks (or threads) is implemented using four types of channels in the programming model:
    1. Asynchronous channels with FIFO semantics (non-destructive write, destructive read). Writing or reading into/from a FIFO (*sc_fifo*) is blocking if it is full/empty.
    2. Register-type channels (destructive write, non-destructive read).
    3. Synchronisation channels (e.g., *sc_semaphore*).
    4. Control signals acting as events (*sc_event*) within the respective modules, notifying (*notify()*) its arrival to intermediate *wait* statements and, consequently, waking them up.
- For each of the different data transfer between threads a user-defined data-type (*struct*) is defined, which includes the single data-elements to be transferred.

- The configuration parameters are defined at the top-level of the system hierarchy and passed to the different sub-modules as constructor arguments.
- Memory allocation cannot be performed explicitly in the code, but is done using requests via specific communication channels. A carefully exploration of data transfers and storage instructions in the original code has to be performed, specially in case of data management through pointers. This is often encountered in multimedia applications which have to deal with huge amounts of data transfers.

The above guidelines form a general framework that is, in principle, applicable to any existing application software. At this point, it is important to remark that neither the language nor the programming model are self-defined, but the novelty of the framework resides on their appliance to develop the architecture-independent multi-threaded model for design space exploration and its subsequent refinement for implementation on a MpSoC platform.

### 4.2. SystemC to Operating System

As introduced in [5], the embedded software to be executed by the processors integrated on a MpSoC can be systematically generated by simply replacing some SystemC library elements by behaviorally equivalent procedures based on operating system (OS) functions. This method is independent of the selected OS (any of them can be supported by writing the corresponding library for that replacement).

The SystemC code for the application is not modified during the software generation flow. All the required modifications are performed at library level and are therefore hidden from the designer. Such a conversion library (SystemC to OS) supports the hierarchy, concurrency, execution control, timed specification, and data types included in SystemC descriptions. For example, the library uses OS API (Application Programming Interface) calls for the concurrency support, so the SystemC threads have to be mapped to the underlying OS threads.

## 5. Mapping and Scheduling onto a MpSoC

Once the architecture-independent multi-threaded model for the application is built, the second stage consist of mapping and scheduling it onto the selected MpSoC. In order to perform a functional validation as well as a system performance estimation, a multi-processor simulation platform is required. At this point we mainly have to distinguish between modelling of task execution on PEs of the architecture and the data exchange between tasks via communication channels mapped to SEs.

For the simulation of task execution on PEs, two different ways can be followed. On one hand, detailed SystemC models of processor microarchitecture executing OS together with the previously generated embedded application software can be used. On the other hand, the SystemC modules representing tasks can be annotated with *wait* statements, which emulate the run time of the corresponding task parts on a PE. The run times are previously estimated using, for example, an instruction set simulator or micro-architecture simulator, such as SimpleScalar [2]. In the current work we have followed the second approach to obtain a rough estimation of system performance.

The modelling of communication channels totally depends on the underlying hardware/OS platform and the mapping of tasks on processing resources. The programming model is not restricted to any particular type of architecture. Some primitives (or communication channels) will be implemented purely in software, whereas some of them will require hardware support. For example, some communication channels will need to be mapped on hardware communication infrastructure such as busses. In this work, we study an emerging paradigm of communication across SoC platforms, the so-called networks on chip (NoC). For building the simulation model, on top of SystemC, the On-Chip Communication Network (OCCN [3]) library has been applied, which facilitates the developing of new models for on-chip communication architectures. It provides an open-source framework for the specification, modelling, simulation and design exploration of NoC. At this point, the structure of the PDU (Packet Data Unit) and bandwidth for each link have to be defined. Furthermore, for interfacing to the packet switched network, the PEs require a wrapper, usually called a network interface (NI), which separates computation from communication.

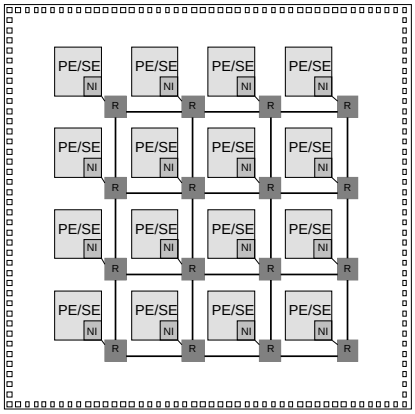


**Figure 1. MpSoC Internal Architecture**

Figure 1 shows an overview of the selected scalable MpSoC architecture, composed of a number of processing (PEs) or storage (SEs) elements connected through a

Network-on-Chip (NoC). This NoC is made up of homogeneous routers organized following a 2D-mesh topology. A processing (PE) or a storage (SE) element is attached to each router (R), which acts as a host sending or receiving packets. A pair of links binds two routers or one router with its respective host, one for each direction.

## 6. Case Study: MPEG-2 Video Decoder

MPEG-2 video decoder has been chosen as a first case study for demonstrating the feasibility of the multi-core solution. Multimedia applications, such as MPEG-2 video decoding, have several interesting properties and pose some challenges for their modelling: They represent a complex streaming application; they can be partitioned onto several concurrent tasks that have variable consumption/production rates and variable execution time; they are known to be resource demanding; finally, the performance of the algorithm, in particular, distribution of load among tasks changes depending on the parameters of a video sequence and the encoded video content.

As a starting point, we took the reference implementation of the MPEG-2 video decoder algorithm provided by the MPEG group [9]. Then, we decomposed the sequential software into a set of parallel independent tasks communicating over a number of channels. The granularity of the decomposition corresponds to the processing on the macroblock level. Thus, the main data exchange between tasks occurs via streams of macroblocks. The motivation behind the chosen granularity of parallelization is twofold. On one hand, there are not so many data dependencies between macroblocks of the same picture. The only restriction that constraints the parallel processing of macroblocks within a single picture is caused by the fact that the VLD task has to access incoming (compressed) bitstream in strict sequential order. There are data dependencies between pictures in the decoded sequence, but these dependencies can be resolved by providing synchronization mechanisms on picture level, which has much coarser granularity. Hence, a large portion of processing that is performed within a picture can be easily parallelized.

Figure 2 shows the selected parallelization of the functions involved in the case study into four main concurrent modules: VLD, IDCT, MC and ADD. The data- and control-flow of the application as well as some SystemC primitives used in the modelling are depicted.

### 6.1. Data Flow

The input of the compressed video bitstream into the system is stored in an input-FIFO. The Variable Length coefficient Decoding (VLD) module starts then extracting and decoding the variable-length coded words from the bitstream

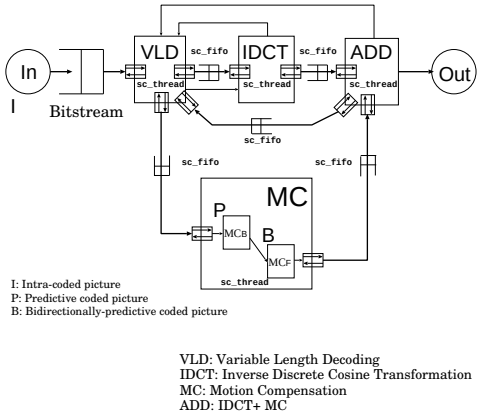


**Figure 2. MPEG2 Video Decoder Data- and Control-Flow**

to obtain motion vectors and quantized values of the DCT (Discrete Cosine Transform) coefficients for each block. This information is then passed to the Inverse Discrete Cosine Transformation (IDCT) module, which reconstructs the pixel values of the actual video sample for I-macroblocks and of the prediction error in case of macroblocks of P and B types. Concurrently, for inter-coded frames (P- or B-macroblocks only), the associated motion vectors and the motion prediction mode are sent to the Motion Compensation (MC) module. MC has an access to a memory where reference frames are stored. It can directly access them, without need to synchronize with other tasks unless the reference frames are updated with newly decoded data. The MC module performs the reconstruction of the frame using the motion vectors. For P-frames, a Motion Compensation Backward ($MC_B$) is performed, and for the B frames, both, a $MC_B$ and a Motion Compensation Forward ($MC_F$) are executed. Lately, the predicted data is added to the prediction error within the ADD module to recover the particular macroblock of the frame. The decoded frames can then undergo a video postprocessing (not shown in the example) and, finally, they can be displayed on the output device.

### 6.2. Control Flow

The coordination of application execution on the platform is implemented via two different event types: (i) events on boolean signals and (ii) events on the FIFOs whenever a data element is written in.

In the case study a boolean signal from VLD triggers the initialization of IDCT; another signal controls the loop between VLD and IDCT for the luminance and chrominance values of one macroblock (it allows a pipeline in the top data-path). Finally, the signal originating from ADD notifies VLD about the end of a macroblock processing.

Data transmission between modules is triggered by events associated to the corresponding FIFOs, which sig-

nalize the writing of new data-elements into the FIFOs (*data_written_event()*).

### 6.3. Mapping onto the MpSoC

The mapping of the previous architecture-independent model for the MPEG2 video decoder into the predefined MpSoC. The MpSoC consists of four processing elements connected by a complex on-chip communication architecture (NoC), is depicted in Figure 3.

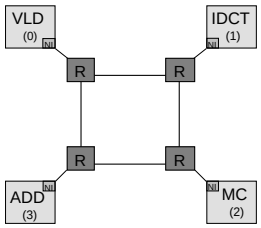


**Figure 3. Mapping of MPEG2 into MpSoC**

The previously implemented SystemC modules for each of the four main tasks are annotated with a *wait* statement, which emulates the run time of the corresponding task on a PE. In addition to it, a network interface (NI) is attached to each module-output for interfacing with the packet switched network (NoC). This NI comprises a host-dependent and a host-independent part. While the first one depends on the host type, the second part can be reused in every host attached to the same NoC. It is responsible for packetization and depacketization of data and data encoding for error detection and correction.

All the elements that made up the communication infrastructure are also modelled in SystemC, using, on top of it, the OCCN [3] library. This enables an homogeneous simulation environment where both, computation and communication, are decoupled but modelled using the same primitives.

For verification purposes, the decompressed video frames resulting from the simulation of the model are compared to the output of the reference software provided by the MPEG group. This is performed for several conformance bitstreams to guarantee the compliance. At this point, the architecture-independent multi-threaded model and its mapping onto the MpSoC infrastructure have been functionally verified. Moreover, the model can be easily applied to explore different mapping alternatives of the respective threads to the software or hardware components of the target architecture.

## 7. Conclusions

This paper has presented a novel multi-threaded programming paradigm that aims to cope with the new challenges introduced by emerging multi-processor system architectures. The proposed framework takes an existing reference software written in C/C++ and, with limited modifications, thread it in SystemC. With the resulting architecture-independent multi-theaded model, one can play out several scenarios and decide the best implementation. Further on, for the threads to be implemented in software, the conversion of SystemC to pure C with pthread library support could be made automatically. And for the implementation of the hardware threads, a refinement of the SystemC model is required, but not a complete translation is necessary.

It has been shown that the system-level language SystemC offers enough primitives for implementing new models of computation more suitable for multi-processing systems. Moreover, it enables the modelling of the architecture, the functionality, and the environment using the same language, which facilitates further simulations and verifications.

## References


[1] Imec conceives 'double-y' methodology for design of multifunctional devices. Press Release, feb 2004.

[2] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modelling. *IEEE Computer*, 35(2):59–67, 2002.

[3] M. Coppola, S. Curaba, M. Grammatikakis, G. Maruccia, and F. Papariello. *On-Chip Communication Network: User Manual V1.0.1*. 2003.

[4] P. Gaulin, C. Pilkington, M. Langevin, E. Bensoudane, K. Szabo, D. Lyonnad, and G. Nicolescu. A multi-processor soc platform and tools for communications applications. *Embedded Systems Handbook, CRC Press*, 2004.

[5] F. Herrera, H. Posadas, P. Sanchez, and E. Villar. Systematic Embedded Software Generation from SystemC. In *Proceedings of International Conference on Design, Automation and Test in Europe*, 2003.

[6] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design*, 19(12), 2000.

[7] S. Kiran, M. N. Jayram, P. Rao, and S. K. Nandy. A Complexity Effective Communication Model for Behavioral Modelling of Signal Processing Application. In *Proceedings of 40 $^{th}$ International Design Automation Conference*, 2003.

[8] B. Lewis and J. B. Daniel. *Multithreaded Programming with Pthreads*. Prentice Hall, 1998.

[9] MPEG homepage. *http://www.mpeg.org/MPEG/index.html*.

[10] J. M. Paul. Programmers' Views of SoCs. In *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis*, 2003.

[11] SystemC homepage. *http://www.systemc.org*.

[12] W. Thies, M. Karczmarek, and S. P. Amarasinghe. StreamIt: A language for streaming applications. In *Computational Complexity*, pages 179–196, 2002.