Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

A model-based software implementation tool for multiprocessor embedded systems

I. Assayad, S. Yovine

Report n^o TR-2005-14

October 2005

A model-based software implementation tool for multiprocessor embedded systems

I. Assayad, S. Yovine

October 2005

Abstract

We apply a formal, automated model-based design tool for synthesizing correct-by-construction parallel implementations of an MPEG-4 video encoder. The tool allows for early prototyping, verification and simulation of embedded applications. The generated software implementations are multi-threaded and customized for system on-chip multi-processor architectures. We consider two HW platforms: a custom industrial video-encoding board currently under development, and Intel's IXP2800 network processor.

Keywords:

Reviewers:

How to cite this report:

@techreport { ,
title = { A model-based software implementation tool for multiprocessor embedded systems},
authors = { I. Assayad, S. Yovine},
institution = { Verimag Technical Report },
number = {TR-2005-14},
year = { },
note = { }
}

1 Introduction

Computation-intensive embedded applications, such as video compression, result in a significant growth in processor workload. To cope with this problem, a solution is to use heterogeneous multiprocessor architectures integrating multiple processor cores, and other specialized hardware components, on a single chip. The software counterpart is that embedded programs become multithreaded to better exploit at software-level the physical parallelism provided by multiprocessor hardware [12].

Design and implementation of multithreaded software for multiprocessor architectures is complex, costly and error-prone. In particular, ensuring non-functional application requirements and platform constraints (e.g., timing properties, resource management, ...) is difficult with today's industrial engineering methods.

Indeed, embedded software engineers must handle several notions of concurrency (e.g., going from multithreading and software pipelinening to dedicated IPs, multiprocessors, and processor-level multithreading), as well as a number of different communication and synchronization mechanisms (e.g., synchronous communication, shared memory, single processor, bus contention, ...)

It has been recognized [1] that there is a need for new analysis and synthesis tools to cope with concurrency, at both software and hardware levels, supported by software development methods based on a unified programmer's model. Such a model should provide (1) appropriate mechanisms for expressing and dealing with different notions and levels of concurrency, and (2) a semantic framework for formally relating them. The ultimate goal is to produce executable code which is correct with respect to application's logic and non-functional requirements.

JAHUEL [2] is a model-based, formal development framework which is being designed along this line of work. It consists of a formal language, called FXML, and its associated tool-suite. FXML provides simple and platform-independent constructs to specify the behavior of the application using an abstract execution model. It has a formal semantics allowing for correctly refining abstract FXML specifications into more concrete ones without semantic break-downs. FXML has been designed to express software structure, functionality, and requirements, and execution-platform architecture and constraints. FXML specifications can be directly manipulated to perform, both platform-independent and dependent, analyses (e.g., schedulability) and transformations (e.g., data and task mappings) to generate code for execution or for simulation. JAHUEL is connected to the FlexCC2 [4] industrial optimizing C-compiler for embedded hardware, and a TLM-based simulation tool for validating the generated code on a simulated platform.

In this paper, we present the results obtained using JAHUEL for synthesizing and evaluating several implementations of an MPEG-4 software for two different platforms. The first one is a custom industrial video-encoding platform currently under development (Sec. 4). The second one is based on Intel's IXP2800 network processor [6] (Sec. 5).

2 The JAHUEL framework

JAHUEL [2] is based on the formal language FXML. In FXML computation units are concurrent by default. Precedence constraints can be used to limit concurrency and to express synchronization and communication. The granularity of computation units is not fixed, the smaller grain is the assignment or legacy code. FXML provides parallel (`par`) and sequential composition, and a `forall` primitive to declare several concurrent iterations of the same block. Basic FXML does not provide any specific synchronization or communication primitives. Instead, the basic language can be extended with non-functional information about and mechanisms of the concrete execution model and target platform (e.g., execution times, synchronization and communication, number of processors, ...)

Compiling an FXML specification consists in transforming it until actual platform-dependent code is generated (either for execution or for simulation). More formally, let $\mathcal{L}_0$ be basic FXML. The compilation chain is a sequence $\mathcal{L}_0 \mapsto^* \mathcal{L}_0 \mapsto \mathcal{L}_1 \mapsto^* \ldots \mathcal{L}_n$, of *correct transformations*. $\mathcal{L}_i$ is an *extension* of $\mathcal{L}_{i-1}$ with primitives not expressible in $\mathcal{L}_{i-1}$. $\mathcal{L}_i \mapsto^* \mathcal{L}_i$ is a sequence of *refinements* (e.g., adding new sequential

[1] See for instance: Platform-based design: A choice, not a panacea, by Richard Goering, EE Times 09/11/2002, http://www.eet.com/reshaping/platformdesign/OEG20020911S0061

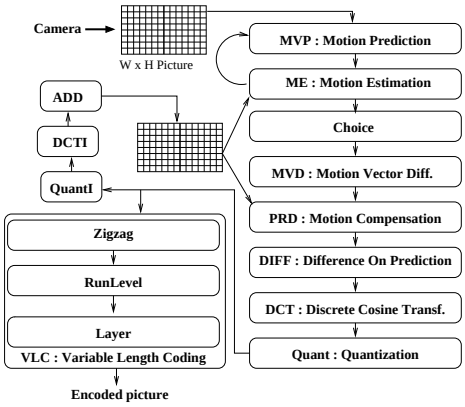


Figure 1: MPEG block diagram view

dependencies), and $\mathcal{L}_i \mapsto \mathcal{L}_{i+1}$ is a transformation that adds information not expressible in $\mathcal{L}_i$ (e.g., the communication and synchronization mechanisms).

The semantics of a specification p of an extension $\mathcal{L}$ of FXML is a set of executions $e \in E_{\mathcal{L}}$. An *execution* is a partial order between computation units (i.e., assignments or blocks of legacy code) which is consistent with data and precedence dependencies. FXML allows expressing dependencies of the form $a \rightarrow_{[l,u]} b$ meaning that every occurrence of b starts in a time $start_b$ such that $start_b - end_a \in [l, u]$, where end_a is the finishing time of a. FXML also allows expressing relationships between *indexed instances* of computations. This is useful for specifying dependencies between iterations of `for` loops or parallel executions of `forall` loops. For instance, $a(i) \rightarrow b(i+1)$ means that the i-th instance of a preceedes the $(i+1)$-th instance of b. Indexed dependencies can be annotated with timing constraints and may quantify over vectors of indexes (for expressing dependencies in nested loops). We will illustrate the use of this construct in Sec. 3 to specify dependencies in an MPEG-4 encoding algorithm.

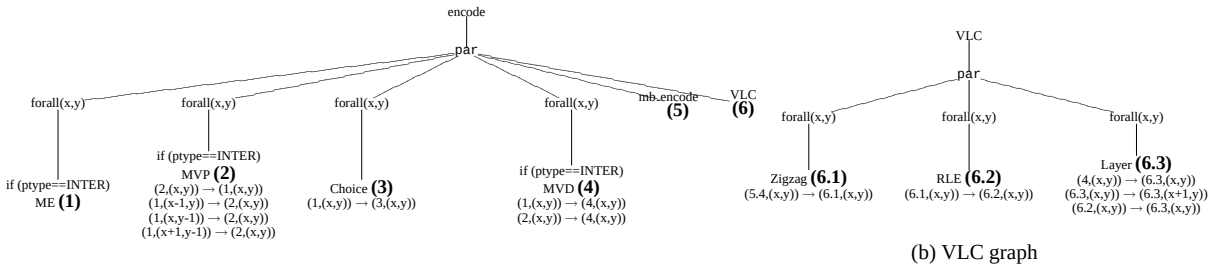
A transformation from $\mathcal{L}$ to $\mathcal{L}'$ is an injective map $\phi : \mathcal{L} \rightarrow \mathcal{L}'$. Let $E_{\mathcal{L}}$ be the set of executions "of type $\mathcal{L}$", and $F_\phi : E_{\mathcal{L}'} \rightarrow E_{\mathcal{L}}$ be the "forgetting" function that forgets all information that is specific to executions "of type $\mathcal{L}'$". A *correct* transformation ϕ is such that for all executions e' of the transformed specification $\phi(p) \in \mathcal{L}'$ we have that $F_\phi(e')$ is an execution of $p \in \mathcal{L}$.

A formal definition of FXML can be found in [2]. Our current prototype provides some general built-in transformations which can be customized for different execution platforms. Indeed, JAHUEL can be easily extended with other transformations and models. For instance, JAHUEL (a) provides code-generation for POSIX-compliant runtime platforms, (b) generates stop-watch automata to enable scheduler synthesis [8, 3], (c) generates code for SystemC-based simulation. The latter allows for early prototyping, verification and simulation. Moreover, automated generation of both executable and simulation code from the same formal model ensures simulation results are trustworthy.

3 Software application: MPEG-4 encoder

In this section we briefly present the specification of MPEG-4 [1] using the FXML language. For lack of space, we only give here a part of the formal FXML model. This model describes all the existing concurrency in the compression algorithm at the macroblock level. Such concurrency does not appear in the MPEG block diagram (Fig. 1).

The specification is composed of `forall` nodes, legacy C-code blocks (MPEG-4 computations), and dependencies of the MPEG phases. Nodes are labeled with numbers in brackets (Fig. 2). We note (1,(x,y)) the computation corresponding to the execution of the ME (motion estimation) phase on the frame macroblock at position (x,y). The arrows indicate dependencies between these computations. There are three types of dependencies : (1) data dependencies resulting from the MPEG-4 standard specification (e.g., in Fig. 2(a), (1,(x,y)) → (3,(x,y)) is a data dependency expressing that the ME phase on macroblock (x,y) must



(a) The encode procedure

(b) VLC graph

Figure 2: Encode and VLC procedure graphs

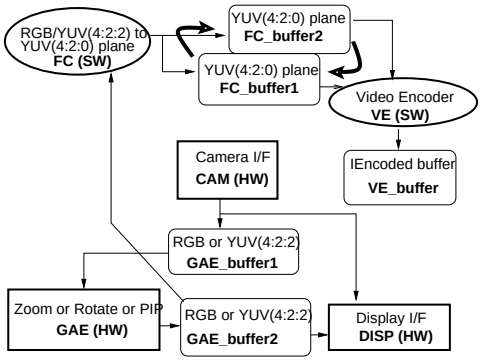


Figure 3: Video-capturing platform

finish before starting the Choice phase on the same macroblock), (2) functional dependencies necessary for the correct functioning of the application (e.g., there is a functional dependency from macroblock (x,y) to macroblock (x+1,y) in the specification of VLC (Fig. 2(b)) because generated headers and blocks are sequentially written in the output bitstream), and (3) dependencies resulting from implementation decisions (e.g., using input and output buffers with one-frame capacity) of encoding frames one after another.

4 Industrial video-capturing platform

4.1 Platform overview

The platform is composed of five components (Fig. 3). Each picture grabbed by the camera (**CAM**) is sent to the (**GAE**) component which is in charge of performing user-requests such as zooming or rotating. Two concurrent components operate on picture data: the display (**DISP**) device and the format-converter software (**FC**). The result of the conversion is used by the MPEG-4 video-encoding software (component **VE**). Software components run concurrently on several processors. All components are periodic processes and communicate throw input and output unit-size buffers located in an external memory. The format-converter software uses two alternating buffers **FC_buffer1** and **FC_buffer2**. This allows **FC** to handle the next frame while **VE** is still encoding the current one stored in the other buffer. Encoded pictures are stored in the **Encoded** buffer.

The hardware test-bed architecture is depicted in Fig. 4. The system has one shared 800 MB/s bus composed of two independent request and a response channels. The bus master's clock frequency is 1.6 Ghz. A simple priority-based bus arbiter is used to grant access to different processors. Transaction size is one word. The bus does not support locking. The system has a 32 MB SDRAM composed of several independent banks of different sizes which can be accessed in parallel. The latency of the memory is 2

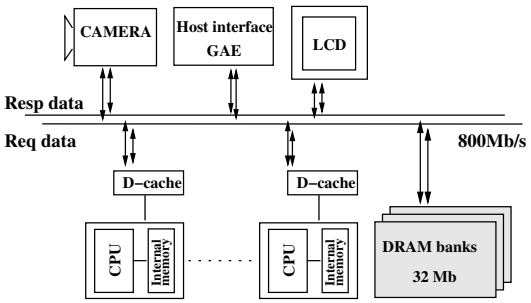


Figure 4: Video-capturing hardware board

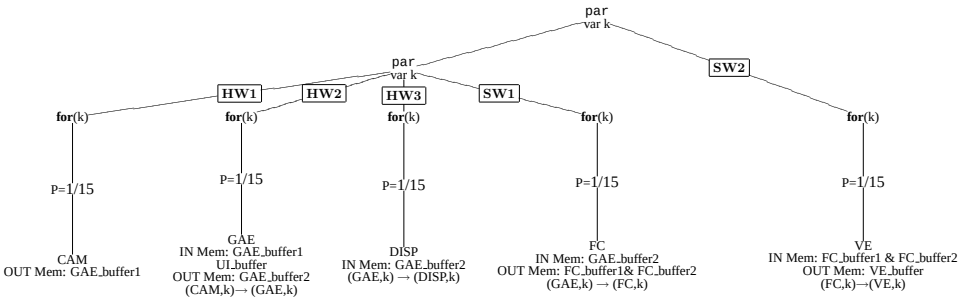


Figure 5: FXML model of the video encoding platform

cycles for a request when no bank conflict occurs. The size of a memory data transaction is one word. In addition to IP components (for capturing and displaying), the board can be populated by up to 18 processing elements for executing the embedded software. Each processor has a 16 Kb data cache with 4 Kb pages size. Caches have a 32 bytes block-size, and are configured for direct-mapped associativity, and use the least-recently-used replacement and on-demand fetch policies with write-allocate strategy. Each processor has a separate OS. The OS supports static memory allocation for making user-defined data mappings, ensures cache coherence, and provides inter-processor synchronization via a POSIX-compliant wait/notify API.

The behavioral properties of the platform are modeled in FXML (Fig. 5). IP components are identified by labelling the corresponding nodes HW1 (CAM), HW2 (GAE) and HW3 (DISP). Software nodes are labelled SW1 (FC) and SW2 (VE). **IN MEM** and **OUT MEM** indicate input/output data buffers. (CAM,k) $\rightarrow$ (GAE,k) indicates that the k-th computation of **GAE** uses data produced by the k-th computation of **CAM**. The FXML model of the MPEG-4 video-encoding software **VE** is the one given in section 3. The composition of both models specifies the behavior, structure, requirements and constraints of the whole application.

4.2 Architecture-specific implementation

4.2.1 Scheduler synthesis

In order to meet the timing requirements of the application, we have to synthesize a feasible scheduler. The scheduler must ensure that software tasks complete within the specified period of 1/15 seconds (i.e., an encoding rate of 15 frames per second). HW execution times are assumed to be known. On the other hand, SW execution times may vary and are therefore treated as parameters.

Our approach consists in synthesizing a scheduler from the FXML model of the application applying the compositional technique developed in [3]. The model only takes into account HW execution times and periodic activations, and interactions among software and hardware components due to dependencies. That is, the model abstracts away from platform-dependent issues (such as cache misses, bus accesses, ...) that

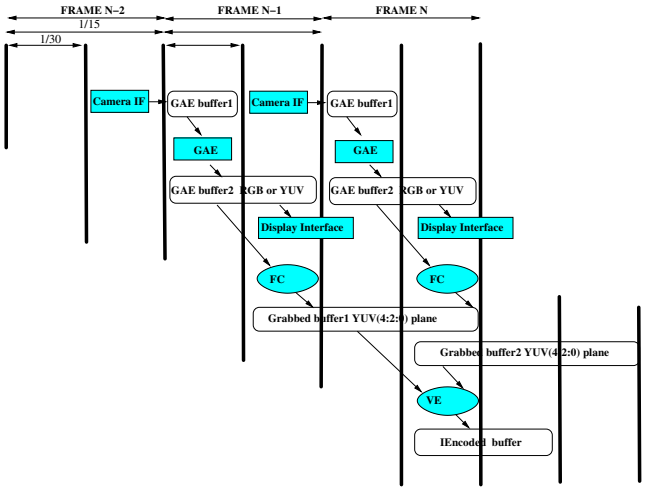


Figure 6: Feasible scheduling

considerably affect SW performance. From this model, the synthesis procedure derives a scheduling and a constraint on the execution-times of **FC** (δ_{FC}) and **VE** (δ_{VE}) which must be satisfied at runtime for the scheduling to be feasible. The constraint is: $\delta_{FC}+\delta_{VE} \leq \frac{1}{15}$ (**Eq.1**). Figure 6 shows a feasible scheduling.

4.2.2 Implementation synthesis

Synthesizing an implementation consists in finding a deployment of processors and SW components that yields SW execution times that ensure the already synthesized constraint (Eq.1). In what follows, we assume that the WCET of **FC** is 1/38 seconds. Replacing in Eq.1 we get that the execution time δ_{VE} has to be: $\delta_{VE} \leq \frac{1}{25}$ (**Eq.2**) to ensure schedulability. A correct parallel implementation must satisfy Eq.2. We use JAHUEL to generate different implementations whose performances are analyzed on several simulated hardware platforms. These implementations take into account platform-dependent issues that have been abstracted away when synthesizing the scheduler. Of particular interest are the behavior of the bus and the memory to study the impact of memory accesses on software execution times. In particular, simulations give information about bandwidth utilization and cache misses, and allow relating them to software performance.

Data-to-memory mapping is done in order to reduce memory latency by avoiding memory-bank conflicts between requests. The critical tasks are the motion estimation (ME), the forward discrete cosine transformation (DCT), the inverse discrete cosine transformation (IDCT) and the entropy encoding (VLC), in this order[2]. Independent banks are assigned to ME, ADD and VLC data. The grabbed picture is split into several chunks by **FC** and also mapped into separate banks.

Table 1 summarizes part of the data-to-memory mapping. For simplicity neither buffer addresses nor address ranges covered by memory-banks are shown. The grabbed picture (GRB) is loaded by **CAM** into a memory bank GRB_bank (W CAM), which is the memory location of **GAE_buffer2** (Fig. 3, to be read by **FC** (R FC). The picture is then split by **FC** and written into different banks GRB[j], where $j = 1, 2$ is one of the two alternating buffers **FC_buffer1** and **FC_buffer2**. Each of these buffers is indeed distributed over several banks according to the number of encoder threads: data for the i-th thread is loaded into memory bank GRB_bank_j_i. And so on. This scheme increases system throughput.

Software-to-processor mapping is done in several steps. The first one consists in transforming the initial FXML model of the encoder in such a way that all functions except VLC are grouped under a `forall`

[2]Execution times of these tasks constitute an average of more than 60% of the overall encoder's execution time.

DATA	SIZE(KB)	MEMORY	ACCESS TYPE
GRB	460	GRB_bank	R FC - W CAM
GRB[j]	230	GRB_bank_j_i	R ME(i) - W FC
ADD[j]	230	ADD_bank_j_i	W ADD(i) - R VLC

Table 1: Data to memory mapping

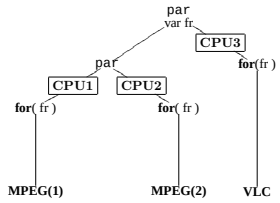


Figure 7: Two-thread implementation

node. For simplicity, we call this group MPEG. This transformation (which is indeed a refinement and therefore allowed by the formal framework) is intended to end-up reducing synchronization overhead at the cost of eliminating some parallelism because of the introduction of new dependencies. The second step consists in partitioning and distributing the tasks according to the target architecture. We have evaluated three strategies. The first one consists in sequentializing the MPEG `foralll` by transforming it into a `for`, and then spawning several copies of it, one into a processor. An instance of this implementation is illustrated in Fig. 7. The second strategy consists in splitting the MPEG `foralll` into an n-stage pipeline. Fig. 8 shows a 4-stage pipeline. The last class of implementations, called hybrid, is obtained by combining the previous two.

We have generated with JAHUEL simulation code for several different instances of these implementations. For this, we have developed a transaction-level model (TLM) of the board on SystemC. Fig. 9 summarizes the simulation results. For lack of space, we show only the results for 4-stage pipelines: (a), (b), (c) and (d) are instances of the first class of implementations; (e), (f) and (g) are hybrid implementations, 2x4, 3x4 and 4x4, respectively.[3] We have used communication buffers with a capacity of 1 macroblock data. The experimental results show that implementations (d), (f) and (g) satisfy Eq.2. Hybrid implementations produce an increase of bandwidth usage due to conflicting parallel requests and the negligable decrease of the rate of cache misses (which stabilizes around 1.3%). The best compromise seems to be implementation (d), consisting of 4 MPEG-threads, which meets the execution time requirement, with an acceptable amount of used bandwidth, using less processors than (f) and (g).

5 IXP2800 NP

The IXP2800 NP [6] is a general purpose 32-bit RISC processor for higher layer network processing tasks. It has 16 RISC micro-engines, connected to all shared resources (SRAM, DRAM, MSF, etc.). IXP2800 NP is composed of three RDRAM channels. Either one, two, or three channels can be enabled. When more

[3] A single 4-stage pipeline did not produce significant results

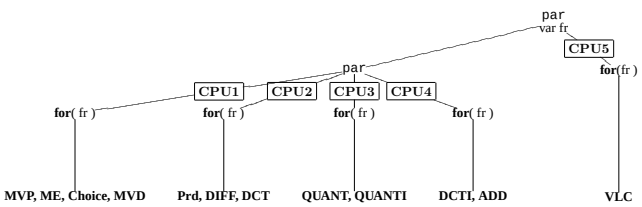


Figure 8: Pipeline implementation

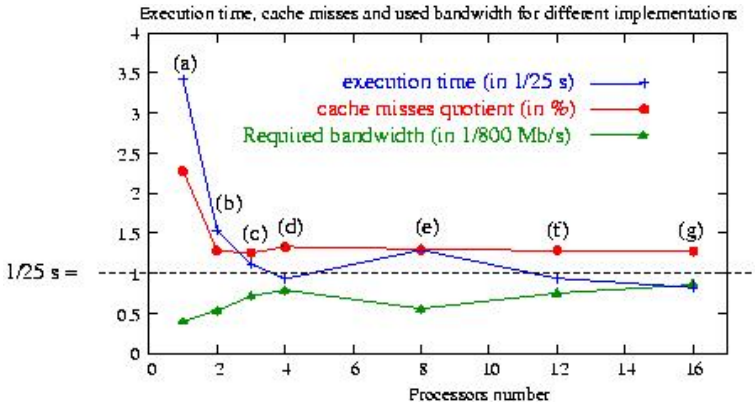


Figure 9: Simulation results

than one channel is enabled, the channels are interleaved on 128-byte boundaries to provide balanced accesses to all populated channels. An address space of 2 GB is supported by the DRAM interface regardless of the number of enabled channels. With interleaving, each channel must have the same number, size, and speed of RDRAMs. Each channel can be populated with up to 32 RDRAMs with the same size and speed. Table 2 gives the used DX Unit (DRAM Unit) characteristics.

Characteristics		Note
Number of channels	1,2 or 3	1, 2 or 3-Way interleaving
Channel frequency	800 MHZ	Round-Robin among DRAM units
Address rearrangement		Based on BANK_REMAP value (=01)
Capacity per channel	32 MB	Delivery rate of 64 B/Cycle
Devices per channel	4	8 MB per device
Banks per device	16	Round-Robin among banks
Bank size	512 K	Closed bank policy
Page size	512 B	1024 page per bank
PULL/PUSH buses	2800 MB/s	4*64-entry command FIFO per channel

Table 2: RDRAM channel characteristics

Channels are interleaved on 128-byte boundaries in hardware to improve concurrency and bandwidth utilization. Within a channel, contiguous 128-byte blocks are directed to non-contiguous RDRAM resources by rearranging physical address bits in a programmable manner.

Within the 3-way interleave mode the channel is selected using modulo-3 reduction (address bits [31:7] are summed as modulo-3, and the remainder is the selected channel number). When two channels are active, address bit 7 is used as the channel select. Addresses that have bit 7 equal to 0 are mapped to channel 0 while those with bit 7 equal to 1 are mapped to channel 1. The address within the channel is [31:8], [6:0]. When only one channel is active, all accesses go to that channel.

In addition to interleaving across different RDRAM channels, addresses are also interleaved across RDRAM chips and internal banks. This improves utilization since certain operations on different banks can be performed concurrently. The interleaving is done based on rearranging the channel address as a function of memory size. The rearranged address is partitioned to choose RDRAM chip, bank within RDRAM, and page within bank.

We have synthesized several implementations of the MPEG-4 encoder for the IXP2800. For lack of space, we only show the simulation results (Tab. 3) obtained for the sixteen MPEG-thread implementation (see Sec. 4), using all three addressing modes. This implementation uses all IXP micro-engines and satisfies Eq.2.

We notice that even if the bandwidth load is fairly distributed among used channels, all modes result in almost the same frame rate and bandwidth usage. This is due to full use of the PUSH bus, which shows that the PUSH bus is the bottleneck that prevents from taking full advantage of the IXP channel and memory

parallelism for the MPEG-4 application.

Interleaving mode	1-Way	2-Way	3-Way
Execution time δ_{VE} (s)	0.0394	0.0395	0.0395
PUSH Bus used bandwidth (MB/s)	2782.3	2781.9	2782.1
PULL Bus used bandwidth (MB/s)	129	129	129
Channel 1 used bandwidth (%)	57.8	26.2	23.3
Channel 2 used bandwidth (%)	0	31.6	15.8
Channel 3 used bandwidth (%)	0	0	18.8

Table 3: Results for 16 MPEG-thread on IXP

6 Conclusions and related work

Other approaches which have also been designed to integrate software and hardware models, and non-functional properties are the following. In architecture description languages (e.g., [5]), the application execution model is tied up to a built-in platform-dependent execution model. Model-integrated development [7] handles horizontally-composed requirements at the same level of abstraction, but it does not seem to be well adapted to reason about cross-cutting requirements that need vertical propagation and composition through different abstraction layers. Platform-based design (PBD) [11] supports vertical integration, but focuses on composing functionality while abstracting away non-functional issues. Metropolis [9] implements PBD and provides simulation, verification and synthesis tools, but currently only supports scheduling policies specified using sequential programs. PTOLEMY II [10] supports composition of heterogeneous models of concurrent computation, but it is oriented towards modeling and simulation rather than to code synthesis.

JAHUEL integrates in the same tool-suite (1) formal model-based, software synthesis, (2) a TLM-based simulation tool for validating synthesized code on simulated platforms, and (3) an advanced optimizing C-compiler for embedded hardware. This enables code generation for specific platforms (including software-to-processor mapping and scheduling), and platform-independent functional analysis, to be linked together in the same tool-chain without semantic gap. We believe the results presented in this paper show that our approach is able to automatically synthesize implementations, satisfying non-functional application requirements and platform constraints, for state-of-the-art hardware platforms and software applications. Future work concerns validating the approach with more industrial applications and extending the tool to handle other constraints such as energy consumption.

References

[1] *Inf. technology – Coding of audio-visual objects – Part 2: Visual*. Prentice Hall, ISO/IEC 14496-2:2001. 3
[2] I. Assayad, V. Bertin, F.-X. Defaut, P. Gerner, O. Quévreux, and S. Yovine. JAHUEL: A formal framework for software synthesis. In *ICFEM'05*, 2005. 1, 2
[3] I. Assayad and S. Yovine. Compositional constraints generation for concurrent real time loops with interdependent iterations. In *I2CS'05*. Springer Verlag, LNCS, 2005. 2, 4.2.1
[4] V. Bertin, J. Daveau, P. Guillaume, T. Lepley, D. Pilat, C. Richard, M. Santana, and T. Thery. Flexcc2: An optimizing retargetable c compiler for dsp proc. In *EMSOFT'02, LNCS 249*, 2002. 1
[5] P. Binns and S. Vestal. Formalizing software architectures for embedded systems. In *EMSOFT'01*, LNCS 2211, 2001. 6
[6] Intel IXP2800 Network Processor Hardware Reference Manual. http://www.intel.com/design/network/manuals/278882.htm 1, 5
[7] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. In *In Proceedings of IEEE,, 91(1)*, 2003. 6
[8] C. Kloukinas and S. Yovine. Synthesis of Safe, QoS Extendible, Application Specific Schedulers for Heterogeneous Real-Time Systems. In *ECRTS'03*, July 2003. 2
[9] Metropolis. Design environment for heterogeneous systems. http://www.gigascale.org/metropolis. 6
[10] PtolemyII. http://ptolemy.eecs.berkeley.edu/ptolemyII. 6
[11] A. Sangiovanni-Vincentelli. Defining platform-based design. In *EEDesign*, February 5 2002. 6
[12] M. Schlett. Trends in embedded-microprocessor design. *IEEE Computer*, 31(8):44–49, 1998. 1