Published in IET Computers & Digital Techniques
Received on 23rd December 2007
Revised on 19th April 2008
doi: 10.1049/iet-cdt:20070168

ISSN 1751-8601

Multiprocessor platform-based design for multimedia

A.C. Ammari[*] *A. Jemai*[†]

Institut National des Sciences Appliquées et de Technologie (INSAT) BP676, 1080 Tunis Cedex, Tunisie
[*]*Unité de recherche en Matériaux Mesures et Applications (MMA) INSAT BP676, 1080 Tunis CEDEX, Tunisie*
[†]*Laboratoire LIP2, Faculté des Sciences de Tunis, 1060 Belvédère Tunis, Tunisie*
E-mail: chiheb.ammari@insat.rnu.tn

Abstract: The computational requirements for embedded applications are increasing exponentially. This complexity, coupled with constantly evolving specifications, has forced designers to consider intrinsically flexible implementations. In this paradigm, the digital system-on-a-chip platform-based design environment for shared memory multiple instructions multiple data architectures (Disydent) is used. Disydent is based on four tools. The distributed process network is a C library for describing Kahn process network (KPN)-based applications. The ASIM0 is a multiprocessor target platform running a micro-kernel. The cycle accurate system simulator is a high-performance cycle accurate simulator, and the user-guided high-level synthesis is a synthesis tool that may be used to enhance the platform with dedicated coprocessors. The main steps of the design flow are KPN modelling, functional validation, design space exploration and temporal validation. The applicability of the Disydent design flow to systems in the multimedia domain is illustrated. The case studied consists in deploying a motion JPEG decoder onto a configurable prototype of a multiprocessor MIPS platform. This study explores both the modelling and mapping stages of the Disydent design flow for an optimal implementation verifying constraints. For this case, the functional constraint consists in achieving a 25 frame-per-second (fps) decoding rate using 50 MHz processors as a non-functional constraint. The sequential decoder implementation does not meet the constraints. To speed up the decoding, different parallel implementations are performed on several target platforms. For more design space exploration, the influence of different scheduling policies, memory cache size and software/hardware mapping are considered.

1 Introduction

Embedded electronic systems are used to carry out specific tasks and are 'embedded' in their environment. This is in contrast to personal computers or supercomputers which have a general purpose and interact with users. Embedded systems typically have strict performance requirements relating to issues, such as latency, throughput, jitter, memory usage and energy consumption [1]. Because of their widespread usage and the critical nature of their performance, the design of embedded systems is both relevant and challenging.

The choice of implementation architecture determines whether designers want to implement a function as a hardware component or as software running on a programmable component. In recent years, computational requirements for embedded applications have been increasing exponentially. This complexity, coupled with constantly evolving specifications, has forced designers to consider intrinsically flexible implementations. For this reason, and because hardware-manufacturing cycles are more expensive, software-based implementation has become more popular [2].

Previously, greater performance for software processing elements was achieved by increasing the clock frequency, exploiting instruction level parallelism, introducing deeper pipelines and carrying out speculative execution. For these techniques, the amount of performance gain achieved for a

given increase in energy consumption has recently become smaller [3]. Since embedded devices are usually constrained by battery life and/or packaging cost, increased power consumption cannot be tolerated.

The alternative is to increase throughput by adding more parallelism to the system in the form of multiple cores. Each processor can run at a relatively low clock frequency and perform a portion of the specified application. Greater parallelism with relatively simple processing elements running at lower frequencies provides increased computational capabilities with higher energy efficiency.

Heterogeneous multiprocessor architectural platforms are gaining prevalence for embedded systems. These platforms feature multiple processing elements, some of which may be customised for specific domains. Deploying applications typical of multimedia domains is difficult because of not only the heterogeneous parallelism in the platforms, but also the performance constraints that characterise these systems.

This study presents the Disydent design flow [4, 5]. This design flow is based on the platform-based design methodology for shared memory multiple instructions multiple data (MIMD) architectures. This work aims to explore the design flow that addresses modelling and mapping challenges, by applying it to embedded systems from multimedia domains. The case study consists of deploying a motion JPEG decoder application [6] onto the ASIM0 multiprocessor MIPS R3000 target platform.

The paper is organised as follows. The next section provides an overview of the platform-based design methodology. Section 3 presents the Disydent design framework. The case study of a multimedia motion JPEG decoder multiprocessor implementation is discussed in Section 4. Section 5 studies aspects and issues of design space exploration and proposes an optimal implementation.

2 Platform-based design methodology

To deal with constantly increasing complexity, safety and time-to-market pressure, embedded system designers are turning to more rigorous design methods. The platform-based design [7] paradigm has been proposed to cope with these difficulties. In this paradigm, a platform is designed with sufficient flexibility to support the implementation of an entire set of products. The product design problem then involves configuring the platform and deciding which parts of the product's functionality are to be implemented by which platform resources. Usually, designers evaluate several configurations before selecting one that meets design goals. This process is known as design space exploration.

2.1 Architecture platforms

In general, platforms are characterised by programmable components. Thus, each platform instance derived from the architecture platform maintains enough flexibility to support an application space that guarantees the production volumes necessary for economically viable manufacturing. The library that defines the platform can contain reconfigurable components. These can be design-time reconfigurable, such as the Tensilica Xtensa processor [8], which allows considerable configuration of the instruction set processor for applications. They can also be run-time reconfigurable via reconfigurable logic, as with field-programmable gate arrays. A designer derives an architecture platform instance from the platform by choosing a set of components from the platform library or by setting the parameters of the library's reconfigurable components. Programmable components guarantee a platform instance's flexibility to support different applications. Software programmability yields a more flexible solution.

2.2 Design issues

From an application domain perspective, performance and size are the constraints that usually determine the architecture platform. For a particular application, a processor must meet a minimum speed and the memory system must meet a minimum size. Because each product has a different set of functions, the constraints identify different architecture platforms; more complex applications yield stronger architectural constraints.

Once an architecture platform has been selected, the design process involves exploring the design space defined by that platform's constraints. These constraints can apply not only to the components themselves, but also to their communication mechanism. Application developers first choose the architectural components they require, yielding a platform instance. Then, they map their application's functions onto the platform instance. This mapping process includes hardware and software partitioning. For example, the designers might decide to move a function from software running on one of the processors to a hardware block, which could be full-custom logic, an application-specific integrated circuit or reconfigurable logic. Once the partitioning and the platform instance selection are finalised, the designer develops the final and optimised version of the application software [9].

2.3 Case of Disydent design flow

The Disydent [4] is an open framework for a system-on-a-chip platform-based design for shared memory MIMD architectures. The platform-based design problem is defined according to the following elements: system, application and constraints. Although the system already exists, the application is not supported by the system.

The application is functionally described as a set of processes exchanging data. The actual implementation of the application is physically realised on the platform. The platform is made of several components, including general-purpose processors (CPU), processors dedicated to special tasks (co-processor or hardware accelerator), and memories to store the application software and data. It also includes on-chip interconnects to connect the processors, co-processors and memories. The platform instance is described in a structural 'VHDL' file. This contains cycle true behavioural models of the hardware components and the configuration used for their interconnect structure and capabilities.

Designing an embedded application starts with a high-level description of the application. This description is either a report in natural language or a program in an executable sequential language. There are also non-functional specifications, usually defined as cost constraint (area of target hardware), time constraint (maximum computing durations) and power consumption constraint. For the application designer, the Disydent top–bottom simulation-based approach helps to go from the high-level description to the actual embedded application implementation verifying the constraints.

3 Disydent design framework

This section presents the Disydent platform-based design framework. First, components and tools are described; then, the associated design flow is discussed.

3.1 Disydent components

The distribution of Disydent contains the following packages. They are presented from higher to lower conceptual levels, which are, in fact, the order of their use.

- Distributed process network (DPN) is a C library implemented on top of the POSIX threads of the FIFO communications of the Kahn model [10] for inter-process communication with bounded storage, blocking reads and blocking writes.

- ASIM0 target platform (Fig. 1): this platform is basically composed of one or more MIPS R3000 processors with instruction and data caches, an interrupt controller, a peripheral interconnect (PI) bus [11], RAM and ROM devices, a FIFO interface and direct memory access controllers. For multiprocessor architectures, the memory coherence is ensured by the use of a write-through cache with bus snooping. The RAM and ROM sizes are parameters that should be adjusted depending on the application. This basic ASIM0 platform can optionally be extended by adding FIFOs and HW coprocessors.

- Cycle accurate system simulator (CASS) allows building a cycle accurate simulator [12] of a given hardware. Once the simulator is generated, one loads it with embedded software and then performs cycle precise simulations of a system (embedded hardware and software).

- An open embedded system: this provides an environment to design and simulate embedded systems. It contains CASS modules that are cycle true behavioural models of the embedded hardware, a MUTEK micro-kernel (basic operating system) to run on the processors of the embedded hardware and tools to generate the embedded software being composed of the embedded MUTEK micro-kernel plus the software dedicated to the embedded system [13].

- User-guided high-level synthesis (UGH) is a synthesis tool [14]. Its inputs are a C or VHDL program and the

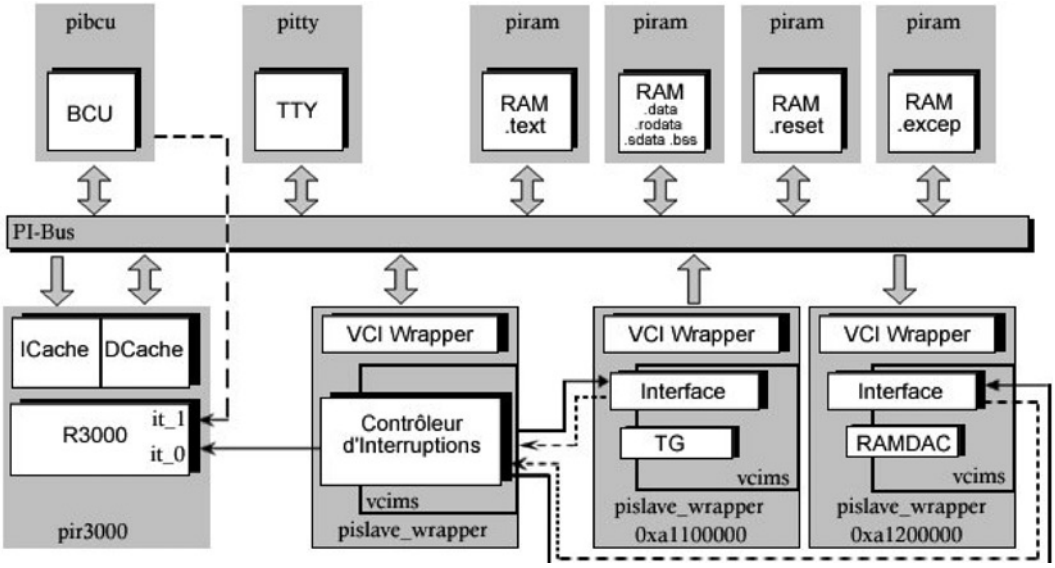

Figure 1 *ASIM0 target platform*

doi: 10.1049/iet-cdt:20070168

clock frequency. It produces both a synthesisable VHDL model and a cycle precise 'C' description that can be used as a coprocessor in CASS simulations.

3.2 Disydent design flow

The current flow for Disydent is typically an adaptation of the uniprocessor design flow. The uniprocessor flow typically involves manual implementation of code for the processor in a low-level language such as assembly code or C followed by extensive simulation to debug and meet performance constraints [15]. The adaptation of this strategy for parallel platforms usually adds an initial partitioning step and parallel specification for the application model in the form of process networks, where concurrently executing processes communicate with each other using explicit messages.

The applicability of the Disydent tools to the multimedia domain comes essentially from the use of the Kahn process network (KPN) [10] model of computation. The KPN model of computation is implemented by the C DPN library. This consists of concurrent processes that communicate with each other through one-way point-to-point FIFOs. Read actions from these FIFOs block until at least one data item becomes available. The write actions block when the FIFOs are full. The execution of a KPN is deterministic and independent of process interleaving. This means that for a given input always the same output is produced and the same workload is generated, irrespective of the execution schedule. The key characteristic of the KPN model is that it specifies an application in terms of distributed control and distributed memory which allows us to map the application onto a multiprocessor platform in a systematic and efficient manner.

For this framework, the hardware of the embedded system is described in a 'VHDL' file. This file represents the description of the platform instance. It contains cycle true behavioural RTL models of the embedded hardware components (MIPS processors, memories, text terminals etc.), and the configuration used for their interconnect structure and capabilities. Once the platform instance is defined, the designer should use the CASS tool to build a cycle accurate simulator for this platform. Next, the software part of the application is cross-compiled and linked with the MUTEK embedded kernel. Finally, the generated binary files are loaded into the target platform simulator.

To transform an application from a sequential specification to an optimal implementation verifying constraints, four main phases are used in the design flow, as shown in Fig. 2. This design flow is not fully automated in that the designer plays a central role. At each phase, Disydent provides information that guides the designer to the appropriate solution. The design flow starts by a sequential implementation of the application on a simulated platform. This step will help obtain profiling information for the next phases. For this, the designer has to modify the initial C description, particularly the input/output (I/O) functions for general-purpose computers, to adapt them to the I/O components of the target platform.

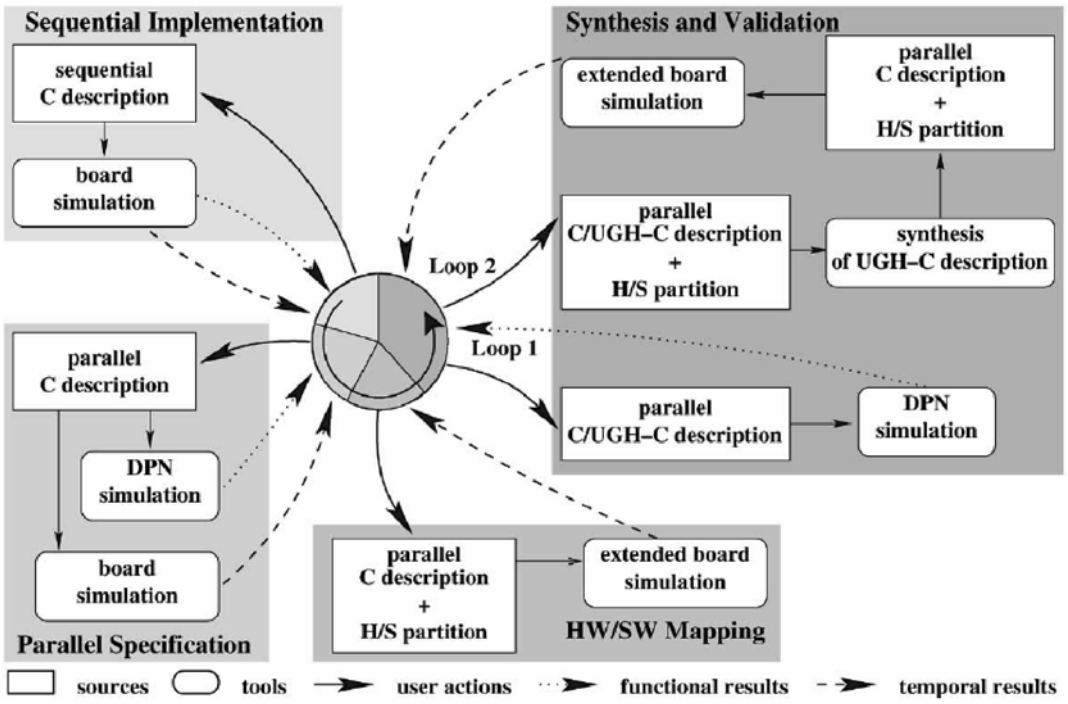

Figure 2 *Disydent design flow*

If constraints are not met, the designer moves up to the next step to implement the parallel specification based on the KPN model. To describe the KPN, the C DPN library implements the KPN communications, with the restriction that the FIFOs have a finite depth. The user defines a C KPN specification of the application using the sequential execution profiling information and his knowledge of the application. The parallelising process is performed by modifying and restructuring manually the original sequential source. There is no tool which can help with this process.

Once functionally validated, the KPN specification is implemented on the multiprocessor platform using the MUTEK multiprocessor multithread kernel. MUTEK provides a standard-based KPN communication API allowing suitable abstraction of the platform. Using this API at this level makes the implementation straightforward and enhances software portability and reuse across different platforms. Mapping the KPN processors to the target processors is managed by the MUTEK kernel using three different scheduling policies. This includes the symmetric multiprocessor (SMP), non-SMP-centralised and non-SMP-distributed schedulers. In the SMP kernel, there is only one scheduler allowing the KPN processes to migrate between processors. The asymmetric centralised scheduler (CS) (non-SMP-centralised) is the only one scheduler statically affecting processes to processors and the asymmetric distributed scheduler (non-SMP-distributed) instantiates one scheduler to every processor and, thus, the processes are statically affected to processors [13].

If the software implementation of the implemented multiprocessor KPN model still does not meet the constraints, then the designer looks for a suitable hardware/software mapping of the application. At this third step, groups of tasks to migrate in hardware must be identified for suitable HW/SW partitions among the processes of the parallel specification. The last step is the synthesis which consists of an automated synthesis of the hardware-mapped processes using the UGH tool [14], and the temporal validation of the final implementation.

4 Motion JPEG decoder case study

In this section, the applicability of the Disydent design flow to systems in the multimedia domain will be illustrated. Multimedia systems deal with computation carried out on streams of data [16]. Data streaming applications such as audio, video and image codecs as well as wireless communication, all characterised as multimedia systems, are predominant in many consumer electronics. The model of computation most often used for these systems is a specialisation of KPN, where actors consume data from input streams, carry out computation and produce data on output streams.

The case study consists in developing a motion JPEG decoder application onto the configurable prototype of the ASIM0 platform architecture. The JPEG decoder is a multimedia application whose building blocks are used in many image and video processing algorithms. In particular, the DCT, quantisation and Huffman blocks are utilised in several other image/video compression applications, including the new generation H.264 standard [17].

For this study, the objective is to explore both the modelling and mapping stages of the flow for an optimal implementation of the MJPEG decoder verifying constraints. The functional constraint consists in achieving a 25 fps decoding rate using 50 MHz processors as a non-functional constraint. Thus, the monitored parameters are representative only of temporal performance aspects such as the thread computing time, bus transfer rate and primitive transfer latency.

4.1 Motion JPEG decoder

The block diagram for the used motion JPEG decoder is shown in Fig. 3. The input for the application is a stream of JPEG images from a traffic generator (TG) input peripheral. The first functional block, DEMUX, dispatches the input stream to the other blocks. VLD performs a Huffman variable length decoding. ZZ reorders the stream of coefficients. IQ performs the inverse quantisation. IDCT performs the inverse discrete cosine transform. LIBU is not a JPEG operation, but it is necessary to adapt the pixel stream to a given RAMDAC peripheral controller output.

4.2 Sequential program implementation

The sequential implementation of the JPEG decoder consists in executing the initial C source application on the simulated ASIM0 platform. For this, we first adapted the I/O original functions to the target platform. We used a TG and a RAMDAC (screen) to replace the I/O files. The CASS simulator has been used to execute the sequential application on the target platforms. We obtained an execution computing time of $n = 103 \times 10^6$ cycles for a 25-frame decoding. At 50 MHz, this represents a duration of 2.05 s. This decoded number of frames per second is $\text{fps} = 25/d = 12.2$. Thus, the sequential implementation does not meet the constraints, and a speedup of 2.6 is needed.

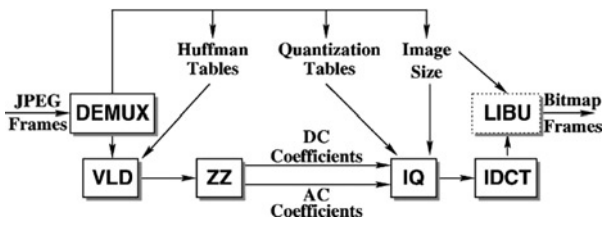

Figure 3 *Motion JPEG figure block diagram*

doi: 10.1049/iet-cdt:20070168

Table 1 Motion JPEG decoder sequential profile

Module	IDCT	VLD	IQ	ZZ	Others
time percentage	67%	13.3%	7.8%	6.1%	5.8%

To achieve the required performance for real-time operation, it is necessary to explore multiple ways of parallelisation. Profiling the execution of the sequential application shall identify the major bottlenecks and the main subcomponents candidate for efficient parallelisation. The obtained profiling results are given in Table 1. These results confirm that the IDCT is the most computationally expensive critical task.

4.3 Parallel specifications

We used three parallel schemes for the motion JPEG decoder algorithm, one is based on pipelining the processing steps as shown in Fig. 4*a*, and the other uses data parallelism to perform the IQ_, ZZ and IDCT in parallel, as shown in Fig. 4*b*. The last model presented in Fig. 4*c* is based on parallel processing of the IDCT as it is the most complex module. For all these models, the macro block level granularity has been used for the communication granularity between tasks.

The implementation of these parallel schemes has been performed by restructuring the original sequential source. This code is modified and structured by hand to describe the KPN in C for varying levels of parallelism. Each Kahn process is described by a C function that has DPN FIFOs as parameters, and inter-process communication is performed using only the DPN I/O primitives. Using global variables for this purpose is not allowed. The proposed high-level parallel model of Fig. 4 has been validated. The correctness of the parallelised code is proved by comparing both execution results of sequential and parallelised codes using the same testbenches.

4.4 Parallel multiprocessor implementation

Once the parallel KPN specifications have been functionally validated, they are cross-compiled with the MUTEK operating system for effective implementation on several

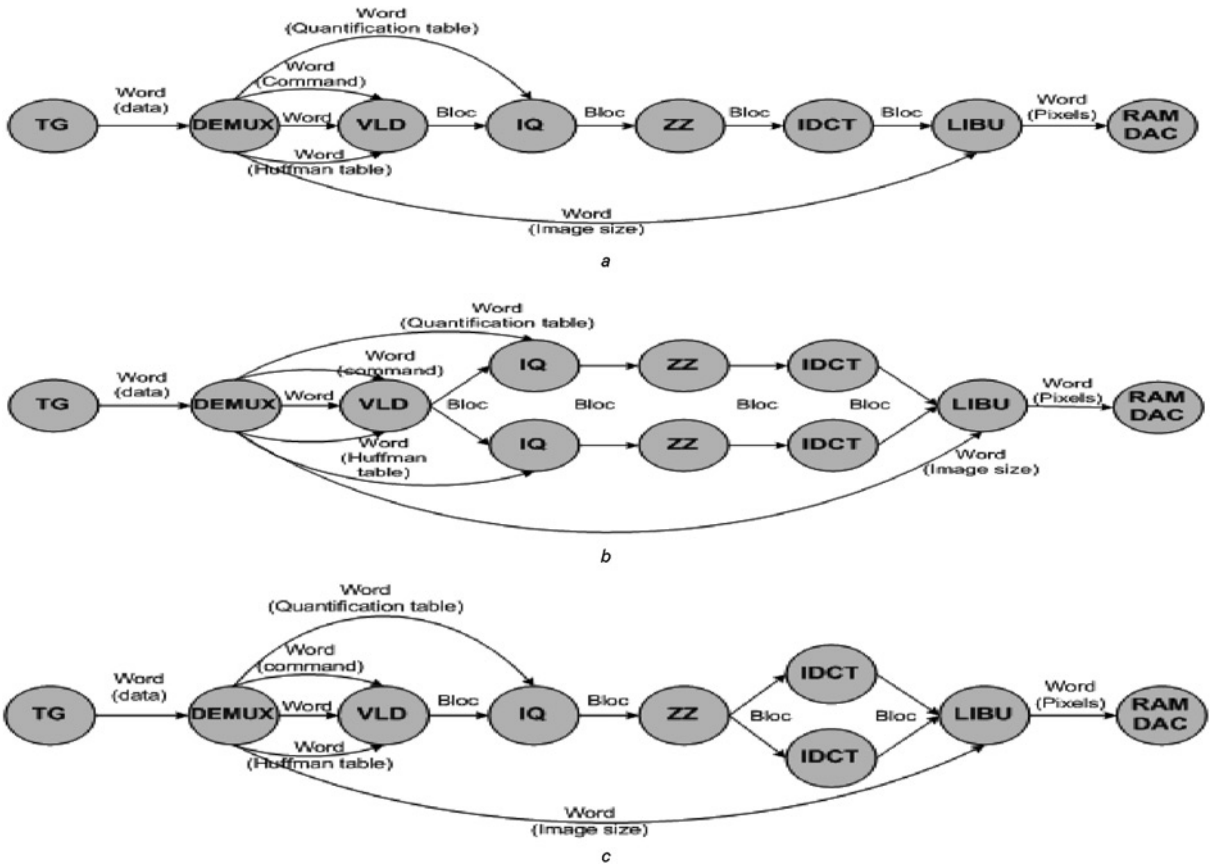

Figure 4 *Motion JPEG decoder parallel specifications*

a KPN pipeline model of motion JPEG decoder
b First parallel scheme
c Second parallel scheme

target platforms. The platforms differ by the number of MIPS processors used. This MUTEK kernel proposes a FIFO-based KPN communication layer. Mapping the KPN processes to the target processors is managed by the MUTEK kernel using various scheduling policies. For this case, we first used the SMP scheduler.

The pipeline KPN model of Fig. 4*a* has been implemented on the ASIM0 target and the execution has been simulated using CASS. ASIM0 is first configured with one MIPS processor, and then enhanced by adding one, two, three and four processors. The MIPS processors have been used with 2 KB cache memory. For each simulation, CASS provides the duration in number of cycles, and so the decoded number of frames per second is calculated. The obtained results are presented in Fig. 5*a*.

It is shown that using only one MIPS processor, the computing time (6.72 fps) is worse than that obtained with the sequential implementation (12.2 fps). This is because of the computing overhead obtained by the inter-process communication. Using two, three and four processors has speeds up the computing, but this acceleration is saturated with the fourth processor to 13.86 fps. Thus, such a model implementation is far from achieving the 25 fps functional constraint.

The parallel implementation of the IQ_, ZZ and IDCT, shown in Fig. 4*b*, has given a notable acceleration. Actually, with five MIPS processors we obtained a speedup of 2 and a decoded number of frames per seconds of 23.35. For this model the performance saturates with five processors. Using the double IDCT model of Fig. 3*c*, we obtained the same performance with five processors; however, with only four processors we obtained 22.09 decoded fps. As reflected in Table 1, the IDCT represents the most complex module. It is anticipated that using more data parallelism for the IDCT will help better verify the functional constraints with minimal processors.

5 Design space exploration

The multiprocessor implementation of the proposed parallel models of Fig. 4 has been performed on several ASIM0 target platforms using several MIPS processors with 2 KB cache memory along with a MUTEK SMP scheduler. The performance obtained did not meet the constraints. For further design space exploration, the influence of the non-SMP-CS along with the memory cache size and software/hardware mapping are considered.

5.1 Memory cache influence

It is demonstrated in [4] that the KPN communications increase access to the bus and consequently, the number of cache misses. A cache miss usually increases the bus-load and induces large system latencies. To see the influence of the memory cache on the decoding performance, we simulated again the execution of the pipeline KPN model of Fig. 4*a* using processors with larger 4 and 32 KB data and instruction caches. The obtained results are shown in Fig. 5*b*.

It is shown from this figure that using 4 KB of memory cache has accelerated the execution by about 5% compared with that obtained with 2 KB. The execution with 32 KB did not give substantial gain with one or two processors. On the contrary, with four processors, we obtained a performance deceleration compared with what had been obtained with 2 KB. This is because of the fact that the more the number of processors increases, the more memory consistency and cache coherency problems [18] will arise. For our platform, the MUTEK kernel uses software solutions to insure cache coherency. The execution time

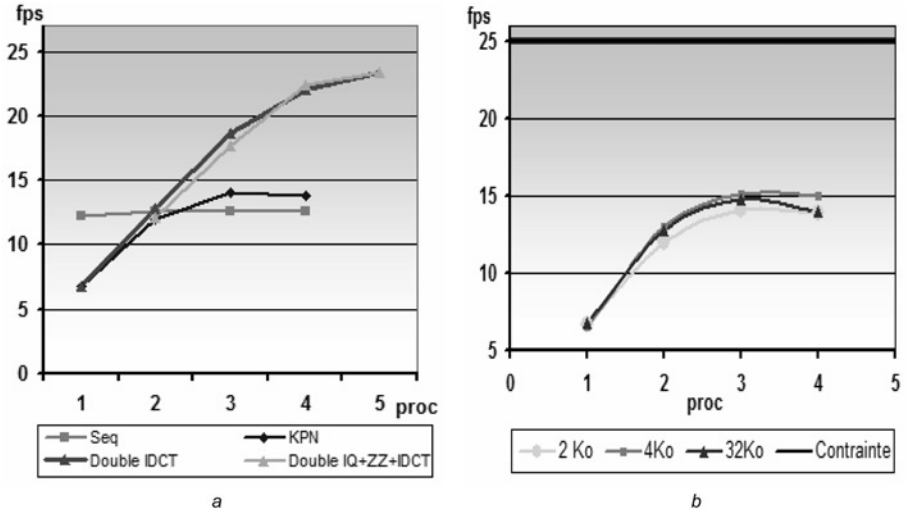

Figure 5 *Multiprocessor performance and memory cache influence*

a Parallel multiprocessor performance
b Pipeline model memory cache influence

doi: 10.1049/iet-cdt:20070168

needed to resolve these coherency problems will cause an added overhead and complexity to the system. Thus, for an optimal implementation a sort of compromise in the size of the memory cache is needed.

5.2 Non-SMP-centralised scheduling influence

Using an SMP kernel, the KPN processors can run on any processor and, thus, may migrate between processors. This allows theoretically a better distribution of the load on all CPUs and a more efficient resource utilisation [10] is obtained. However, the task migration has a high cost in terms of cache misses. Thus, the interest in the SMP might become less for large data exchange applications as is the case of the motion JPEG decoder.

For the JPEG decoder, the IDCT process is the most computationally expensive process. Hence, it is anticipated that using the centralised non-SMP-CS scheduler with dedicated processors computing the IDCT may give a potential decoding acceleration. The double IDCT parallel execution of Fig. 4*c* has been evaluated with three and four processors using, respectively, the mapping scenarios of Figs. 6*a* and 6*b*. Three processors are used for the case of Fig. 6*a* with the IDCT2 and LIBU statically mapped to processor P3, IDCT1 and ZZ to processor P2 and the rest to processor P1. The fixed scenario with four processors is shown in Fig. 6*b*.

The obtained performance results of these implementations are presented in Fig. 7*a*. This figure clearly indicates that the static non-SMP scheduler did not give too much acceleration. For the same number of processors, the performance obtained is comparable with the dynamic SMP case. To obtain a better idea about the efficiency of each scheduling policy, we reported in Fig. 7*b* the processor cycles spent in the idle loop. It appears from Fig. 7*b* that the SMP kernel spends much more less time in idle loops than the non-SMP-centralised. This outlines the capacity of a dynamic SMP scheduler to use

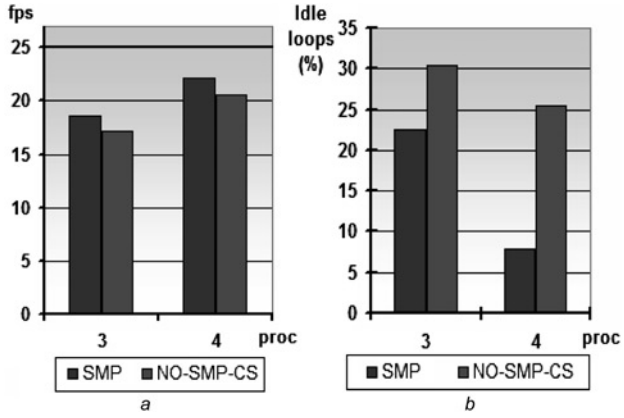

Figure 7 *Obtained performance results of the implementations are presented*

a Non-SMP-centralised/SMP performance
b Processor cycles spent in Idle loop

the CPU cycles more efficiently. However, for the SMP scheduler, the task migration caused a very high cost in terms of cache misses leading to less interest in the SMP policy. Finally, for these cases, the obtained performances with the SMP and non-SMP strategies are comparable. Nevertheless, to more effectively put the accent on performance improvement with static non-SMP scheduling for large data exchange applications as in the case of the motion JPEG decoder, a better static mapping scenario should have been considered with better utilisation of the CPU cycles of each processor used.

5.3 Hardware/software mapping

Hardware-manufacturing cycles are more expensive and so prior to opting for hardware design, it is necessary to be sure that it will be useful. The problem is to find one or more groups of processes that are good candidates for HW mapping. To estimate the appropriate selection, Disydent proposes an exploratory migration approach. To obtain a system with processes

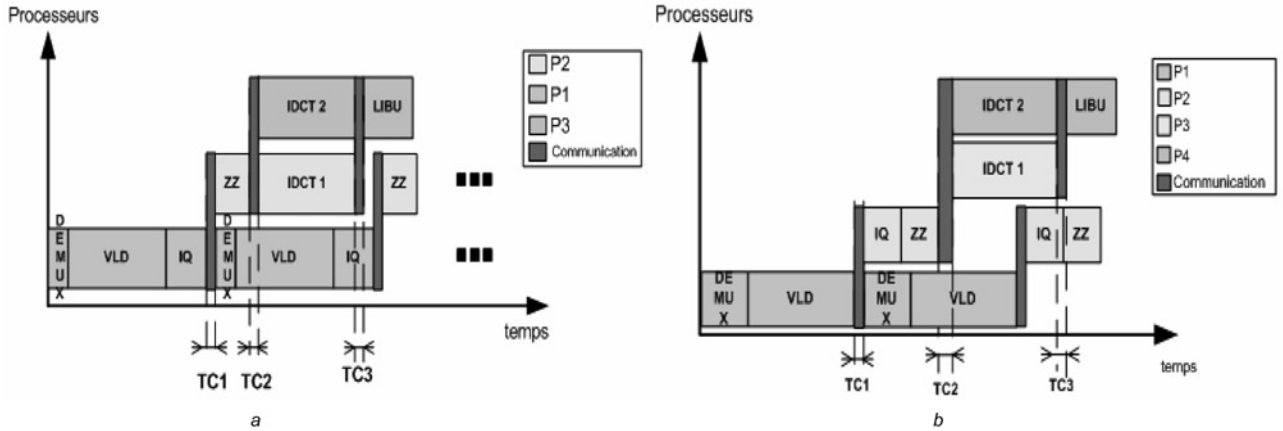

Figure 6 *Static centralised scheduling scenario with*

a Three processors
b Four processors

Table 2 Implementation performance of the decoder MJPEG

Proc	SMP-CS				Non SMP-CS		SMP-CS		
	Sequential	KPN	ZZ + IQ + IDCT	Double IDCT	ZZ + IQ + IDCT	Double IDCT	IDCT hard	IDCT + VLD hard	IDCT + VLD + IQ hard
1	12.6	6.72	–	6.78	–	–	7.7	17	24.12
2	12.6	12.17	12.16	12.86	12.68	–	11.49	24.07	32.45
3	12.6	14.12	17.65	18.63	–	17.19	11.55	36.73	32.56
4	12.6	13.86	22.35	22.09	–	20.56	–	–	–

migrated to HW, the designer has to select the appropriate candidate processes for HW implementation and perform some modifications on the platform description and on the software parallel specification [4].

The selection of the appropriate processes for HW implementation is performed using solely profiling results and intuition of fitness of a particular function for HW implementation. For our case, the profiling results presented in Table 1 indicate that the IDCT is the most computationally expensive critical task. The IDCT process should, thus, be hardwired. The VLD and IQ processes are also good candidates. Therefore three process groups have been selected for hardware implementation: (IDCT), (IDCT + VLD) and (IDCT + VLD + IQ).

Hardware implementation of the (IDCT) has realised 11.55 fps with three processors on the platform. By mapping the group (IDCT + VLD) into hardware, the constraint was almost satisfied using two processors on the platform (24.07 fps). In this case, a code optimisation and a suitable configuration of the platform are necessary to reach 25 fps. The (IDCT + VLD + IQ) group had given as execution performance 24.12 fps with only one processor on the platform. To conclude, there are two potential solutions: (IDCT + VLD) with two processors and (IDCT + VLD + IQ) with one processor. For these design solutions, the final decision will depend more on area than on delay optimisation.

5.4 Result synthesis

With a memory cache size of 4 KB, the design space exploration synthesis using the SMP, non-SMP scheduling policies and the software/hardware mapping are presented in Table 2. Using Table 2, the optimal architecture will use two MIPS processors, RAM memory, PI-bus and an interruption controller. The MIPS processors are dedicated to execute DEMUX, IQ, ZZ and LIBU processes with an SMP scheduling policy. In addition, an input peripheral is needed for reading video stream (TG) and an output peripheral (RAMDAC) for displaying. To achieve the 25 fps functional constraint, hardware implementation of the VLD and IDCT processes is necessary.

6 Conclusions

The platform-based design paradigm has been proposed to cope with the constantly increasing embedded system design complexity, safety and time-to-market pressure. In this paradigm, the Disydent is used. In this paper, the applicability of the Disydent design flow to systems in the multimedia domain has been illustrated. This comes essentially from the use of the KPN model of computation to express the behaviour of such systems. The case study consists in deploying a motion JPEG decoder application onto the configurable prototype of a multiprocessor MIPS platform architecture.

The motion JPEG decoder is a multimedia application whose building blocks are used in many image and video processing algorithms. This study has explored both the modelling and mapping stages of the Disydent design flow with an optimal implementation for verifying constraints. The functional constraint consists in achieving a 25 fps decoding rate using 50 MHz processors as a non-functional constraint. The sequential decoder execution did not meet the constraints. To speed up the decoding, different parallel implementations have been performed on several target platforms. We first tried the MUTEK kernel SMP dynamic scheduling using MIPS processors with 2 KB of memory cache. For more design space exploration, the influence of the static non-SMP-CS, the memory cache size and the software/hardware mapping have been considered. This has given an optimal architecture for the decoder multiprocessor implementation.

In summary, we find the Disydent design approach efficient. The input is a C program using parallel KPN model primitives. The platform instance is described at the RTL level in an appropriate 'VHDL' file. The temporal validation on the target is straightforward and obtained very fast. This is one to two orders of magnitude faster than classical RTL-level simulators [12]. The functional validation at the KPN level is sufficient to ensure the functionality of the application on the target platform. In addition, the integrated MUTEK operating system provides a standard-based KPN communication API allowing suitable abstraction of the platform. Using this API makes

doi: 10.1049/iet-cdt:20070168

the implementation straightforward and enhances software portability and reuse across different platforms.

Nevertheless, Disydent design flow is not a fully automated approach in that the designer plays a central role in each phase. Particularly, the definition of the system for a new platform is a somewhat complicated task. This requires the design of the needed hardware components at the RTL level and the definition of the corresponding CASS files for cycle accurate simulation. In addition, the version of the MUTEK micro-kernel used is ported only for an R3000 MIPS processor limiting the ways of featuring multiple processing elements and targeting heterogeneous platform architectures. To enhance the use of Disydent for heterogeneous platforms, the micro-kernel has to be further ported on other multiple specialised cores.

7 References

[1] DAVARE A., DENSMORE D., MEYEROWITZ T., *ET AL.*: 'A next-generation framework for platform-based design'. Design and Verification Conf. (DV-CON), San Jose, CA, 21–23 February 2007

[2] CASPI P., SANGIOVANNI-VINCENTELLI A.: 'Guidelines for a graduate curriculum on embedded software and systems', *ACM Trans. Embedded Comput. Syst.*, 2005, **4**, (3), pp. 587–611

[3] OLUKOTUN K., HAMMOND L.: 'The future of microprocessors', *ACM Queue*, 2005, **3**, (7), pp. 26–34

[4] AUGÉ I., PÉTROT F., DONNET F., GOMEZ P.: 'Plateform based design from parallel C specification', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2005, **24**, (12), pp. 1811–1826

[5] SMIRI K., MOALLA M., HARBEGUE H., JEMAI A., AMMARI A.C.: 'Kahn based performance model within a co-design flow'. ESM'2007, 22–24 October 2007, St. Julian's, Malta

[6] JPEG committee: 'Standardized in ISO/IEC IS 10918-1/2'. http://www.jpeg.org/

[7] KEUTZER K., MALIK S., NEWTON A.R., RABAEY J., SANGIOVANNI-VINCENTELLI A.: 'System level design: orthogonolization of concerns and platform-based design', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2000, **19**, (12), pp. 1523–1543

[8] www.tensilica.com/products/x7_processor_generator.htm

[9] SANGIOVANNI-VINCENTELLI A., MARTIN G.: 'Platform-based design and software design methodology for embedded systems', *IEEE Des. Test Comput.*, 2001, **18**, (6), pp. 23–33

[10] KAHN G.: 'The semantics of a simple language for parallel programming'. Proc. IFIP Congress 74, North-Holland Publishing Co, 1974

[11] NIEDERMEIN T., *ET AL.*: 'Draft standard OMI 324: PI-bus'. Technical Report, Open microprocessor initiative, December 1996, Rev. 0.3d

[12] PÉTROT F., HOMMAIS D., GREINER A.: 'Cycle precise core based hardware/software system simulation with predictable event propagation'. Proc. 23rd Euromicro Conf., Budapest, Hungary, September 1997, pp. 182–187

[13] SENOUCI B., BOUCHHIMA A., ROUSSEAU F., PETROT F., JERRAYA A.: 'Fast prototyping of POSIX based applications on a multiprocessor SoC architecture: hardware-dependent software oriented approach'. 17th IEEE Int. Workshop on Rapid System Prototyping (RSP'06), 2006, pp. 69–75

[14] AUGÉ I., BAWA R.K., GUERRIER P., GREINER A., JACOMME L., PÉTROT F.: 'User guided high level synthesis'. VLSI: Integrated Systems Very Large Scale Integration, Brazil, August 1997, pp. 464–475

[15] WOLF W.: 'Computers as components: principles of embedded computing system design' (Morgan Kaufmann, 2005)

[16] TSANG T., LAI R.: 'Specifying multimedia QoS parameters and synchronization using time-estelle', *Int. J. Comput. Internet Manage.*, 2005, **13**, (3), pp. 11–32

[17] KRICHENE H., AMMARI A.C., JEMAI A., ABID M.: 'Performance/complexity analysis of a H264 video encoder'. International Review on Computers and Software(IRECOS), July 2007

[18] PÉTROT F., GOMEZ P.: 'Lightweight implementation of the POSIX threads API for an on-chip MIPS multiprocessor with VCI interconnect'. Proc. DATE'03, 2003, vol. 2, pp. 51–56