Exploring Design Space of Parallel Realizations: MPEG-2 Decoder Case Study

Basant K. Dwivedi, Jan Hoogerbrugge*, Paul Stravers* and M. Balakrishnan
Indian Institute of Technology Delhi, New Delhi, India
{basant, mbala}@cse.iitd.ernet.in
*Philips Research, Eindhoven, The Netherlands
{jan.hoogerbrugge, paulus.stravers}@philips.com

ABSTRACT

Many applications lend them to parallelism at different levels of granularity. We first identify the key issues involved in creating a parallel model of an application. These are done with a view to estimate performance and explore the "parallel" design space to select a suitable design point. The framework presented provides an opportunity to perform this exploration both in the target architecture independent and target architecture dependent manner. An MPEG-2 decoder model in YAPI has been presented which has more parallelism and improved performance. This model has further been mapped onto SpaceCAKE architecture to study its architectural parameters. Detailed results obtained with YAPI simulation (target architecture independent) and TSS simulation (after process-processor binding) on MPEG-2 decoder application establish the effectiveness of our approach.

Keywords

MPEG-2 Decoder, YAPI, Parallel realization, Process, Thread, FIFO

1. INTRODUCTION

Recent advances in network and microprocessor technology have made it possible to introduce a new set of applications and services. High Definition TV (HDTV), Broadcast Satellite Service, Video-conferencing, Interactive Storage Media etc. are few typical examples. These applications need huge amount of data processing both in video and audio domain. The high data rates involved in the applications make computation very time consuming.

Designers mainly follow two architectural approaches for signal processing systems, dedicated and programmable. Dedicated architectures target an algorithm or a set of algorithms. These architectures fully exploit the computational features of algorithm and VLSI implementations of dedicated architectures are optimized for area, power and performance. A good overview of architectural approaches for signal processing systems has been given in [1]. Though dedicated architectures provide good performance, they lack flexibility to further extend the algorithm set.

On the other hand the programmable approach offers a number of advantages. Programmable solutions offer greater flexibility, as the target algorithm set can further be extended by applying proper software modifications. Since a large number of applications can run on the same hardware, per application hardware cost is reduced. On the other hand, as computational properties of algorithms are not fully exploited, it requires that both the architecture and application models be faster. Architecture can be made faster by employing multiprocessing strategy. Hence a parallel model of application running over a multiprocessor architecture offers one of the potential solutions.

The computational resources of a multiprocessor architecture can be exploited fully only when the application model has sufficient parallelism. Parallelism present in signal processing applications can be made explicit using models based on Kahn process networks [2]. Several variants of this model have been reported including [3] and [4]. The drawback of models based on Kahn process network is that they cannot model reactiveness. This limitation is overcome in control flow models such as Statecharts, Esterel and Polis. Once a parallel model of the application is built, significant speed up can be achieved when application runs over a multiprocessor architecture.

The main objective of this work is to identify several modeling issues involved in creating a parallel model using YAPI [4], which can further be used as a starting point to expand the application set and estimate the performance of other application models. An MPEG-2 decoder has been modeled with increased parallelism in the model and improved performance. This model is later used to study the architectural parameters of SpaceCAKE architecture [5].

The remainder of this paper is organized as follows. In Section 2 we introduce YAPI, which we used to model MPEG-2 decoder. Section 3 discusses several modeling issues. Section 4 discusses MPEG-2 decoder model in YAPI and describes how performance is improved. Section 5 describes the experimental environment. Section 6 presents results of the experiments with conclusions in Section 7. In this paper the terms *process* and *thread* are used interchangeably.

2. YAPI

YAPI (Y-Chart Application Programmer's Interface) is a C++ library with a set of rules which can be used to model signal processing applications as process networks. This process network is a variant of Kahn process network.

A process network is composed of a number of process networks, processes and FIFOs. In a process network, a process represents a computing station and a FIFO represents a communication channel. Since a process network can further be part of another process network, an hierarchy of process networks can be created. Figure 1 shows a simple process network, where two processes A and B communicate over FIFO F.

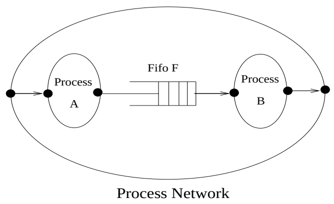

Figure 1: Process Network

Processes define the functionality of the application. A process interacts with its environment through input and output ports. A process can be in *running state*, *blocked state* or *ready state*. A process gets blocked when it tries to read from an empty FIFO or it tries to write into a FIFO which is full.

There are three primitives provided by YAPI for communication over channels. Primitive *read* is used to read from the channel, primitive *write* is used to write into the channel and primitive *select* is used in non deterministic applications. Furthermore, multiple tokens can be transferred over the FIFO using vector communication mechanism of YAPI, which is more efficient. An application model created using YAPI is architecture independent and hence, the application model can be mapped on any architecture.

3. MODELING ISSUES

A parallel model of application is quite different from its sequential model. The state space of a parallel model is quite large. Parallel models also impose overheads of context switching, communication and synchronization. Further, prediction of performance of such parallel application is difficult, as execution speed is highly architecture dependent [6]. In this section, several modeling issues and their effect on performance have been discussed. The focus is at coarse grain parallelism in the context of YAPI.

Amount of Parallelism. The amount of parallelism present in the application depends on application itself. Thorough data dependency analysis of application is required to fully extract the data and functional parallelism present in the application. For example, in MPEG-2 decoder application, several independent tasks such as variable length decoding, motion compensation, IDCT etc. can be identified and data parallelism present at slice, macroblock or block level can be exploited by providing more instances of these tasks. It is very likely that an application model, having more parallelism, will improve performance, provided total number of tokens communicated among processes do not increase much.

Unidirectional Communication. An application model where processes are handshaking with each other (request & grant of tokens between two processes), may lead to a deadlock. Handshaking also increases number of tokens transferred and violates the streaming behavior, where data flows in one direction.

An application model, which has unidirectional communication among processes, is likely to give better performance. This improvement comes from two sources, reduction in number of context switches and number of tokens being communicated. Since data flows only in forward direction in unidirectional communication, it also reduces synchronization overhead and debugging time.

Communication Overhead. In a multiprocessor environment, there is conflict for communication resources (Though with lots of buses and other hardware it is possible to avoid contention altogether, it is expensive). As the number of processors increases in the architecture, communication becomes more expensive. So an application model, optimized for reduced communication, will give better speed. This can be achieved by carefully choosing token structure and controlling the number of tokens transferred over communication channels.

Granularity of Operation. Granularity of operation significantly affects structure of application model. Compilers are there which are able to extract instruction level parallelism at a very fine level of granularity. Here we mainly concentrate on coarse grained parallelism.

In signal processing applications parallelizing at finer granularity may have the following effects:

- Data parallelism present at finer granularity can be exploited.
- Communication primitives are called more frequently, as less number of tokens are transferred during any transfer. So communication workload increases.
- It is more likely that total amount of data transfer also goes up.

Hence, at finer granularity, data parallelism can be increased, but it also increases communication workload. So performance improves only when computation workload relatively decreases.

Balanced Pipelines. In a process network, processes communicate with each other in a pipelined manner. The slowest process in the pipeline determines its throughput. If the computation is not properly balanced among processes, it will reduce effective parallelism present within the application model.

Figure 2 shows two pipelines which converge at process *Sink*. If the latency of lower path is more, because of finite space in FIFOs, processes in the upper path will get blocked more often. This will increase the number of context switches and reduce the effective parallelism in the application. Hence, in such a situation, an application model will perform better which has balanced pipelines.

Synchronization Overhead. In YAPI an application is modeled as a process network. Processes communicate with

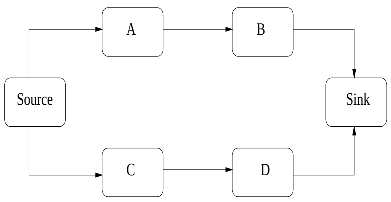

Figure 2: A process network demonstrating two pipelines

each other over FIFOs by transferring tokens. Then it becomes important at what place a process sends or receives tokens. Further, there is need for mechanisms using which global events can be notified and actions can be started. For example, arrival of new picture in MPEG-2 decoding is a kind of global event. All the processes within MPEG-2 decoder model should be notified about this, so that they can update picture properties required for correct decoding. So, the application programmer has to take care of the following:

- Identification of the points at which, a token required for synchronization should be sent.
- Mechanism of notifying processes about global events. One solution is to employ *broadcast mechanism*, in which case some special tokens are communicated to all the relevant processes.
- Mechanism of communicating (sourcing or sinking) tokens with the processes repeating same operation (e.g., IDCT).

Multiple Instance of Processes. The process network in Figure 3 has 'n' instances of process *P*. In this process network, throughput at this stage of process *P* should increase roughly by a factor 'n'. However, this scheme increases synchronization overhead. Now the process *Source* must adopt some policy to distribute the tokens properly among processes *Pi* and similar policy must be adopted by the process *Sink* to collect tokens.

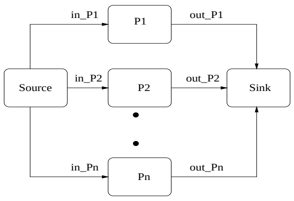

Figure 3: A process network demonstrating multiple instance of process P

If 'n' becomes large, processes *Source* and *Sink* may not respond to demands of processes *Pi* quickly, which may further lead to increase in context switching. So for a large value of 'n' performance improvement may get saturated.

Memory Usage. On chip memory speeds up system performance, but it is limited in amount. This suggests memory usage should be optimized. Increase in parallelism increases number of processes and communication channels within the application model. This further increases memory requirements. So there is a trade off between size of parallelism and space requirements. Though this cannot be eliminated fully, the problem can be reduced by fine tuning the sizes of FIFOs after few experiments.

Scalability. Scalability of the model defines its capability to grow. An application model is scalable if more data parallelism can be provided by just plugging more instances of processes (e.g. by providing more IDCT processes in MPEG-2 decoder application).

4. MPEG-2 DECODER MODEL

A number of parallel models of MPEG-2 decoder [7, 8] in software have been studied [9, 10, 11, 12, 13, 14]. Focus of [9] is on methodology. The focus in [10] and [11] is on extracting parallelism at different levels of granularity (e.g. GOP, slice, macroblock) and studying variation in performance. Whereas, [12] explores the effectiveness of MAJC[1] when a parallel MPEG-2 decoder runs on it. In this paper, we further extend this kind of study.

Based on issues discussed in the previous section, we modeled MPEG-2 decoder (Figure 4) in YAPI. There are three levels of hierarchy in the model. At the top level, process **Tinput** reads input video sequence, process **Thdr** extracts header from the sequence, process **TmemMan** manages frame memory, process **Toutput** outputs decoded frames and process network **TsliceDec** extracts the slice header and decodes the slices.

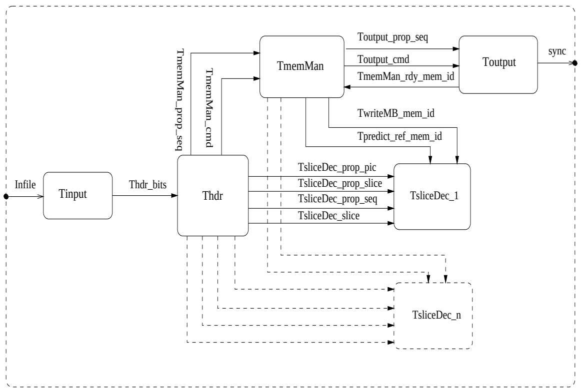

Figure 4: Process network MPEG-2 Decoder

The process network **TsliceDec** (Figure 5) gets slices from the bit stream and decodes it. The process **Tvld** is for variable length decoding. The process network **Tmc** provides motion compensation. Inverse quantization followed by IDCT is provided by the process network **Tiq_idct_add**, whereas the process **TwriteMB** writes decoded macroblocks into the frame memory. Processes at this level operate on macroblocks.

The process network **Tiq_idct_add** (Figure 6) is composed of three processes. The process **Tisiq** provides inverse scan and inverse quantization. The process **Tidct** provides IDCT operation and the process **Tadd** adds IDCT component with prediction component.

[1]Microprocessor Architecture for JAVA Computing

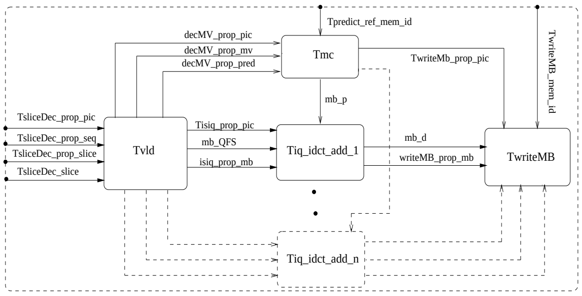

Figure 5: Process network TsliceDec

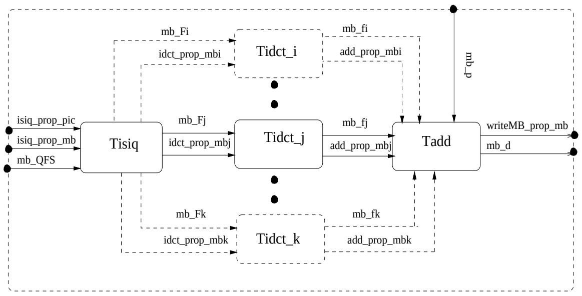

Figure 6: Process network Tiq_idct_add

The processes **TdecMV** which decodes motion vectors and **Tpredict** which provides predictions, make the process network **Tmc** (Figure 7) and provide motion compensation.

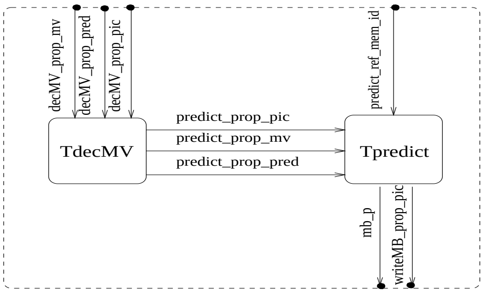

Figure 7: Process network Tmc

The MPEG-2 decoder model described in Figure 4 extracts data parallelism present at slice and macroblock level simultaneously. The model has flexibility to grow. More instances of *TsliceDec*, *Tiq_idct_add* or *Tidct* can be provided to increase parallelism in the application. There is no bidirectional communication except between *TmemMan* and *Toutput*. This cannot be avoided because of limited frame memory. All the FIFOs are resized to utilize FIFO memory space more efficiently.

5. EXPERIMENTAL SETUP

The simulation setup is composed of 4 layers (Figure 8). Uppermost layer is application. Application has been modeled using primitives of YAPI. Since a thread is created corresponding to every process in application model, YAPI uses thread scheduler to provide thread related services.

Thread scheduler is responsible for thread management activities such as context switches, maintaining status of threads etc. The bottom layer is architecture. Every CPU within the architecture has its own thread scheduler. The thread scheduler is like a small operating system which manages the resources of the CPU.

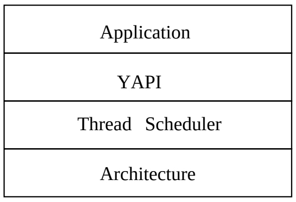

Figure 8: Layered structure of experiment environment

We simulated MPEG-2 decoder application in two environments. These are YAPI run time and TSS environments. In the YAPI run time environment, the application runs on a workstation and YAPI provides application workload information. We simulated MPEG-2 decoder application in YAPI environment on Sun platform with SunOS 5.7.

The other environment is TSS[2]. Here we mapped MPEG-2 decoder application onto TSS model of the SpaceCAKE architecture [5]. The SpaceCAKE Architecture is an homogeneous multiprocessor architecture. The basic unit of repetition is a tile. A tile consists of a heterogeneous mix of memories, general purpose processors (like the MIPS 1900 or ARM), DSPs etc. In our study we used an early version of the architecture, consisting of a single tile with a configurable number of low-end MIPS CPUs, which communicate over a PI Bus. The architecture of these CPUs is very close to PR1910 [15].

We compiled the application model for MIPS and linked it with YAPI library to generate executable of the application. This finally runs on the TSS model of the SpaceCAKE architecture under the control of the thread scheduler. Thus, this process describes the mapping of YAPI model of application onto the TSS model of architecture.

6. EXPERIMENTAL RESULTS

We have taken a number of MPEG-2 decoder models. The first model is named as *old_model* which has been introduced in [9]. Rest of the models can be identified as *mpeg_ijk*. Here 'i' indicates number of process networks *TsliceDec*, 'j' indicates number of process network *Tiq_idct_add* within *TsliceDec* and 'k' indicates number of processes *Tidct* within *Tiq_idct_add*. There are 15 processes within *old_model*. The number of processes within model *mpeg_ijk* can be calculated as follows:

$$Number\ of\ processes\ =\ 4 + i \times (4 + j \times (2 + k)) \quad (1)$$

The input MPEG-2 video sequence is *tennis.m2v* (table tennis sequence with 8 frames, frame size 576x704).

6.1 YAPI Level Simulation

YAPI run time environment is not a multiprocessor environment, as the application runs on a single processor. However, it gives some important feedback about the application. This feedback is in terms of two parameters, *Number of context switching* which is the total number of thread

[2]TSS is a cycle accurate C language based simulation framework used within Philips.

switching on the processor and *Parallelism number* which is the average number of processes in the ready list of the processor at any time. Table 1 shows these two parameters. It can be seen that *Number of context switching* comes down by increasing the value of 'i' in models *mpeg_ijk*, but increases when j goes from 2 to 4. This implies that it is quite possible that after a certain point by providing more instances of processes or process networks will increase *Number of context switches* even if *Parallelism number* increases.

Model name	Number of context switching	Parallelism number
old_model	346763	3.87
mpeg_122	21786	7.81
mpeg_222	20506	13.58
mpeg_242	31826	22.89
mpeg_422	19919	23.87
mpeg_442	31965	39.66

Table 1: Number of context switching and Parallelism number for various MPEG-2 decoder models

6.2 TSS Level Simulation

We have used three instances of architecture, with 2 CPUs, 4 CPUs and 8 CPUs. Various MPEG-2 decoder configurations run on TSS model of SpaceCAKE architecture. We present simulation results corresponding to total number of cycles, CPI (cycles per instruction), bus wait cycles and total number of snooping requests.

Figure 9 shows Number of cycles against number of CPUs for various models. Total number of cycles for different configurations have been normalized against total number of cycles taken during simulation of *old_model* with 8 CPUs. Except *old_model*, other models show decrease in number of cycles with increasing number of processors in the architecture. More than 100% improvement in speed can be seen for *mpeg_222* compared to *old_model*. However, the gain in speed, while going from 4 CPUs to 8 CPUs, is less compared to gain from 2 CPUs to 4 CPUs.

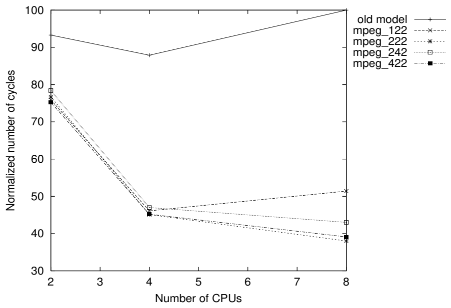

Figure 9: Number of cycles vs Number of CPUs

The *old_model* contains quite a few processes handshaking with each other. Though the *old_model* seems to offer enough parallelism for at least 2 CPU configuration, communication is increased because of handshaking (bidirectional communication). This is the reason why new models which have handshaking only at one place offer significant performance improvement. Further, in the new models, variation in performance increases when there are more processors. Hence, in this case having more processes in the application model offers advantage.

CPI is the ratio of total number of cycles over total number of instructions, taken by the application to decode *tennis.m2v* during TSS simulation. In Figure 10, except old_model all other models show increase in CPI, which indicates decrease in total number of instructions relative to total number of cycles.

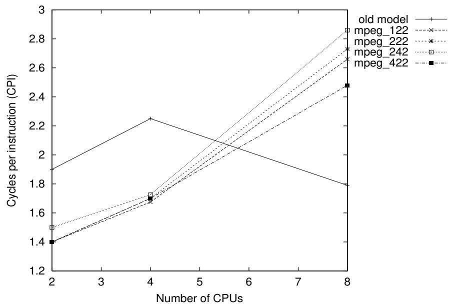

Figure 10: CPI vs Number of CPUs

Figure 11 shows variation of bus_wait_cycles for different configurations. In this plot bus_wait_cycles indicates the average number of cycles a CPU is blocked while a cache fill request is in progress. Except old_model all other models show the same kind of behavior and variation is not very high. The curve is nearly a straight line. So increase in bus_wait_cycles is almost 9 units per processor, whereas old_model show slightly different behavior and increase in bus_wait_cycles is 6.4 units per processor. The reason for increase in bus_wait_cycles is that now there are more CPUs to share the only communication resource (bus).

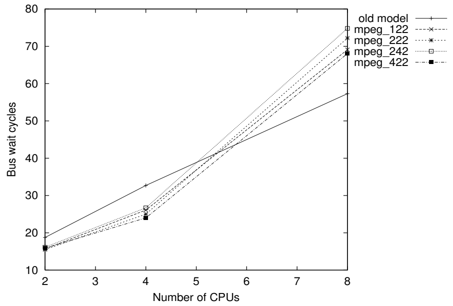

Figure 11: Bus wait cycles vs Number of CPUs

Figure 12 shows the total number of snooping requests made by CPUs for cache coherence. The TSS model of the CPU uses MSI cache coherence protocol for cache coherence among CPUs. It is clear from the plot that as the number of CPUs is increased, required snooping requests also go up. The curves are again nearly straight lines. Increase in snooping request is approximately 3.83 units per CPU for old_model and 1.1 units per CPU for others. Hence relative decrease, in number of snooping requests in new models compared to old_model, is by a factor of 3.4, which indicates there is relatively much less cache coherence activity compared to old_model.

In Figure 13 *Parallelism number* have been taken from YAPI level simulations. It can be seen that variation in the total number of cycles taken during the TSS simulations is not large for a particular number of CPUs. This implies that ideally total number of cycles taken during TSS simulations should decrease as *Parallelism number* increases, but the communication bottleneck prevents this.

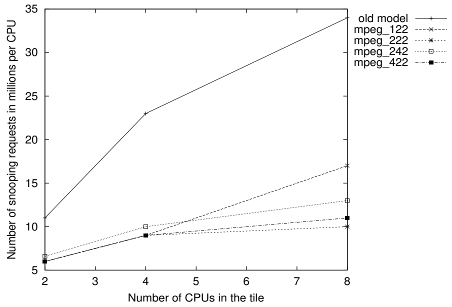

Figure 12: Number of bus snooping requests vs Number of CPUs

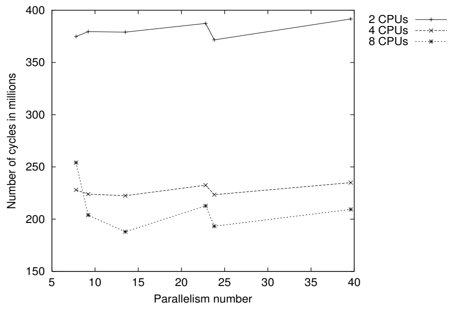

Figure 13: Number of cycles vs Parallelism

7. CONCLUSIONS

There are a number of factors which directly affect the performance of an application model. Unidirectional communication, balanced pipelines, granularity of operation are a few typical examples. The model features such as bidirectional communication, unbalanced pipelines, feedback loops etc. could reduce the speed significantly and might lead to deadlocks.

We found from experiments that as the number of processors in the architecture increases, conflicts for communication resources increases. So after a certain number of processors, increasing the number of CPUs does not improve performance significantly. We also observed that for our MPEG-2 decoder application, presence of 4 CPUs in the tile gives performance which is good enough, as going beyond 4 CPU does not improve performance significantly. Further, adding more CPUs must be backed up with more communication bandwidth. We also observed that for fixed number of CPUs in the architecture, there is a limit beyond which increasing the number of processes in the application model does not improve performance much.

In the current implementation of MPEG-2 decoder model, we allowed only process *Tinput* to get access to video sequence directly. Rest of the processes get access using buffering mechanism. The model can further be improved by reducing buffering and allowing more processes (particularly variable length decoders) to have direct access to video sequence as proposed in [10]. On the other hand this scheme is likely to increase the synchronization overhead.

8. REFERENCES

[1] Peter Pirsch and H. J. Stolberg. VLSI implementations of image and video multimedia processing systems. *IEEE Trans. on Circuits and Systems for Video Technology*, 8(7):878–891, November 1998.

[2] Gills Kahn. The semantics of a simple language for parallel programming. In *Proc. IFIP Congress 74*. North Holland Publishing Co, 1974.

[3] Edward A. Lee et al. Dataflow process networks. *Proc. of IEEE*, 83(5):773–801, May 1995.

[4] E. A. de Kock et al. YAPI: application modeling for signal processing systems. In *Proc. 37th Design Automation Conference (DAC'00)*, June 2000.

[5] Paul Stravers and Jan Hoogerbrugge. Homogeneous multiprocessing and the future of silicon design paradigms. In *Proc. International Symposium on VLSI Technology, Systems, and Applications (VLSI-TSA 2001)*, April 2001.

[6] David B. Skillicorn and Domenico Talia. Models and languages for parallel computation. *ACM Computing Surveys*, 30(2):123–169, June 1998.

[7] Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg and Didier J. LeGall. *MPEG VIDEO COMPRESSION STANDARD*. Chapman & Hall, New York, 1996.

[8] *Information technology - Generic coding of moving pictures and associated audio information: Video.* ISO/IEC 13818-2, 1996.

[9] Pieter van der Wolf et al. An mpeg-2 decoder case study as a driver for a system level design methodology. In *Proc. CODESS'99*, 1999.

[10] E. Iwata and K. Olukotun. Exploiting coarse-grain parallelism in the mpeg-2 algorithm. Technical Report CSL-TR-98-771, Stanford University Computer Systems Laboratory, September 1998.

[11] Angelos Bilas et al. Real-time parallel mpeg-2 decoding in software. In *Proc. 11th International Parallel Processing Symposium (IPPS)*, April 1997.

[12] MAJC Documentation. *MPEG-2 Video Decompression on a Multi-processing VLIW Microprocessor*. http://www.sun.com/microelectronics/MAJC/.

[13] H. Oehring et al. Mpeg-2 video decompression on simultaneous multithreaded multimedia processors. In *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques (PACT '99)*, October 1999.

[14] Brian C. Smith Ketan Patel and Lawrence A. Rowe. Performance of a software mpeg video decoder. In *Proc. ACM Multimedia Conference*, 1993.

[15] *PR1910 User Manual*. Philips Semiconductors.