Efficient multicore scheduling of dataflow process networks

Hervé Yviquel, Emmanuel Casseau, Matthieu Wipliez, Mickaël Raulet

EFFICIENT MULTICORE SCHEDULING OF DATAFLOW PROCESS NETWORKS

Hervé Yviquel, Emmanuel Casseau

IRISA, University of Rennes 1,
6 rue de Kerampont,
22300 Lannion, France.
{firstname.lastname}@irisa.fr

Matthieu Wipliez, Mickaël Raulet

IETR, INSA of Rennes,
20 avenue des buttes de Coësmes,
35200 Rennes, France.
{firstname.lastname}@insa-rennes.fr

ABSTRACT

Although multi-core processors are now available everywhere, few applications are able to truly exploit their multiprocessing capabilities. Dataflow programming attempts to solve this problem by expressing explicit parallelism within an application. In this paper, we describe two scheduling strategies for executing a dataflow program on a single-core processor. We also describe an extension of these strategies on multi-core architectures using distributed schedulers and lock-free communications. We show the efficiency of these scheduling strategies on MPEG-4 Simple Profile and MPEG-4 Advanced Video Coding decoders.

Index Terms— Dataflow computing, Multicore processing, Scheduling algorithm, Distributed algorithm, Lock-free multithreading

1. INTRODUCTION

Since processor frequency is bounded due to physics constraints like power dissipation, multi-core architectures have become the solution to allow performance to keep growing as described by Moore's law. These architectures present an interesting challenge: produce applications which fully exploit the parallelism provided by these processors. Several programming languages, extensions and models allow parallelism to be expressed in applications like Occam [1], Message Passing Interface [2] or Algorithmic Skeletons [3]. Most of them assume a specific underlying hardware architecture and are generally inefficient on other platforms.

Dataflow programming is an attractive candidate to design parallel applications in an architecture-agnostic way. A dataflow program is composed of atomic processing blocks that communicate with each other with communication channels. Such a representation explicitly describes task-level parallelism within the application. The behavior of this kind of program is governed by a Model of Computation (MoC) which specifies a set of rules regarding the execution of the program. Several dataflow MoCs exist with different purposes and expressivenesses such as Synchronous Dataflow (SDF), Kahn Process Network (KPN) [9] and Dataflow Process Network (DPN) [8].

Dynamic dataflow models like KPN and DPN are very useful to describe the behavior of streaming applications. Contrary to SDF and similar models that consume and produce data in a static way, KPN and DPN models may have a data-dependent behavior i.e. the quantity of data consumed and produced by the processes may depend on the value of these data.

This paper proposes several scheduling techniques to efficiently execute dynamic dataflow programs on single-core and multi-core processors. Indeed, determining a schedule of a dynamic dataflow program is not possible at compile-time (equivalent to the halting problem, see [4]). This paper makes the following contributions:

- We give a formal definition of a round-robin policy for scheduling dataflow process networks on single-core architectures that has been successfully used in practice, see [5] and [6] (Section 3.1).
- We present an efficient strategy to dynamically schedule actors of dataflow process networks on single-core architectures (Section 3.2). A clever scheduling strategy is key to ensure that a complex application with many processing blocks can be executed efficiently.
- We propose a distributed and lock-free extension of these two dynamic scheduling strategies on multi-core architecture (Section 4) based on Lamport's work [7] concerning lock-free communication channels of distributed processes.

This paper presents results obtained with our scheduling strategies for the execution of two video decoders (MPEG-4 Simple Profile and MPEG-4 Advanced Video Coding) on a multi-core processor.

2. BACKGROUND

This section presents the dynamic dataflow model called dataflow process network and how applications described by

this model are scheduled.

2.1. Dataflow Process Networks

A Dataflow Process Network (DPN) is a model of computation [8] which can be described as follow: a set of processes called actors that communicate with unidirectional and unbounded FIFO channels called *data-fifos* and connected to ports of actors. A *data-fifo* has only one source port but may have several target ports. In the dataflow approach, the communication between actors corresponds to a stream of data composed of a list of tokens. Figure 1 presents a network of five actors linked by *data-fifos*.

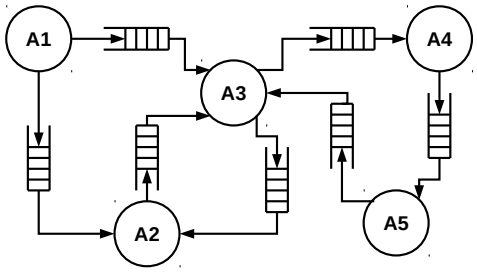


Fig. 1: An example of Dataflow Process Network

DPNs can be considered as a generalization of the well-known Kahn Process Networks (KPNs) [9]. The execution of an actor corresponds to the mapping of input tokens to output tokens applied repeatedly and sequentially on one or more data streams. This mapping is composed of three ordered steps: data reading, then computational procedure, and finally data writing. These repeated mappings are called actor firings and are guarded by a set of firing rules which specifies when an actor can be fired. These firing rules specifies precisely the number and the values of tokens that must be available on the input ports to fire the actor. This is why DPNs can describe nondeterministic algorithms which is not possible with the KPN model.

2.2. Scheduling of Dataflow Process Networks

DPNs are more suitable than KPNs to be scheduled on processors when there are more processes than processor cores because no context has to be saved between each actor execution. Indeed, the execution of an actor is described by a sequence of actor firings and actor switching can only happen between two firings, so only state variables need to be saved; in particular, it is not necessary to save the execution stack and other contextual information like registers, etc.

Consequently, it is possible to reduce the overhead of scheduling a dynamic dataflow program by using a user-level scheduler rather than relying on threads scheduled by the operating system kernel. Von Platen shows an impressive acceleration of 3.4 to 105 frame per second (FPS) with a video decoder after using a user-level scheduler that uses a single thread to schedule actors in [6]. Scheduling an actor with this method is a lot more efficient than using threads because there is no need for the kernel to perform a context switch each time an actor is scheduled but only a function call.

The *data-fifos* used throughout this paper are statically bounded to be implemented on finite memory and avoiding additional overhead due to dynamic memory allocation.

3. SINGLE-CORE SCHEDULING STRATEGIES

A dynamic dataflow program cannot be fully scheduled by static methods so some dedicated strategies need to be developed in order to schedule actors during the execution. The following strategies are designed to execute a DPN-based application on a single-core architecture which handles the execution of only one actor at a given time.

3.1. Round-robin scheduling strategy

Round-robin is a simple scheduling strategy that continuously goes over the list of actors: The scheduler evaluates the firing rules of each actor, executes the actor if a rule is met and continues to execute the same actor until no firing rules are met. This scheduling policy guarantees to each actor an equal chance of being executed, and avoids deadlock and starvation. Contrary to classical round-robin scheduling, there is no notion of time slice: an actor is executed until it cannot fire anymore in order to minimize the number of actor switching and consequently the scheduling overhead. The reason of this unfireability is that *data-fifos* will be finally full or empty because of their bounded sizes.

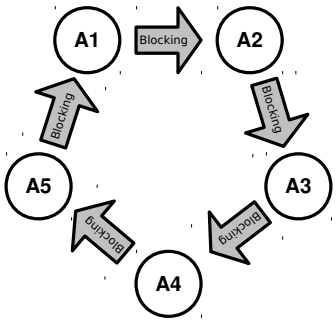


Fig. 2: Example of round-robin scheduling with five actors

Figure 2 shows an application of this round-robin scheduling on the example presented in Fig. 1. The scheduler executes the actors in a circular order i.e. the five actors A1, A2, A3, A4 and A5 are successively executed then the scheduler starts again from A1 and so on.

3.2. Data-driven / demand-driven scheduling strategy

Data-driven / demand-driven strategy is a more advanced scheduling strategy of dynamic dataflow programs. Indeed, the round-robin strategy schedules actors unconditionally i.e. the firing rules of an actor could be checked even if they are all invalid. The firing rules of the actor will be checked, but no computation will be performed. As a result, the round-robin strategy becomes inefficient with complex applications containing many actors and a lot of control communications.

Data-driven / demand-driven scheduling strategy is based on the well-known data driven and demand driven principles [10]. On the one hand, data-driven policy executes an actor when its input data have to be consumed to unblock the execution of the precedent actor. On the other hand, demand-driven executes an actor when its output is needed by another actor.

Two types of events can cause the blocking of an actor execution, each one is implying a different scheduling decision:

- When an actor is blocked because an input *data-fifo* is empty, demand-driven policy is applied and the scheduler executes the predecessor of this *data-fifo*.
- When an actor is blocked because an output *data-fifo* is full, data-driven policy is applied and the scheduler executes the successors of this *data-fifo*. Indeed a *data-fifo* can be connected to several target ports (see Section 2.1).

Contrary to the round-robin algorithm, a dynamic list of next schedulable actors is needed. The behavior of this schedulable list is illustrated with Fig. 3. When an actor is blocked during its execution, the empty or full *data-fifos* are identified and their associate predecessors or successors are added to the schedulable list. The actor to be executed next corresponds to the next entry in the schedulable list.

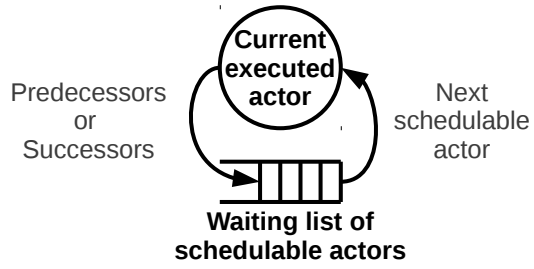


Fig. 3: Example of data-driven / demand-driven scheduling

4. DISTRIBUTED AND LOCK-FREE MULTI-CORE SCHEDULING

This section describes a distributed and lock-free multi-core scheduling technique to execute dynamic dataflow programs using round-robin and data-driven / demand-driven strategies.

4.1. Distributed scheduler

A distributed scheduler is designed to execute applications on a multi-core architecture. Several local schedulers are executed concurrently on each processor core. This specific design avoids the use of a specific thread to manage the scheduling of the application.

A static partitioning of the actors on the processor cores is needed to run our multi-core scheduler. On the one hand, the round-robin strategy goes over a static list of actors so its multi-core extension needs this static mapping of the actors on the cores to be implemented. On the other hand, the data-driven / demand-driven strategy could work with a mapping computed dynamically, but (1) a static mapping allowed us to develop the multi-core extension by tackling each problem one at a time, and (2) is considered as future work. Figure 4 presents a possible mapping of five actors on a dual-core processor.

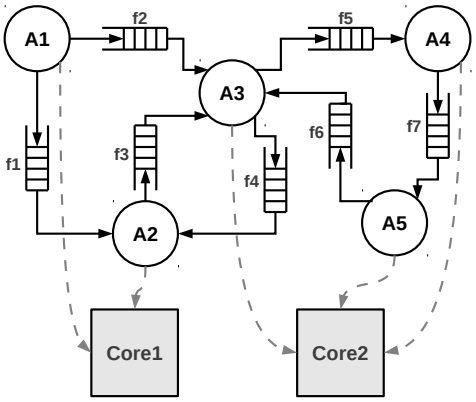


Fig. 4: Mapping example of a network on processor cores

To form the distributed scheduler, one thread for each available processor core is created and forced to be run only on its associated core. The round-robin algorithm executes a subset of the actors that are mapped on the associated core in each thread. Figure 5 shows an example of the distributed scheduler with the round-robin strategy.

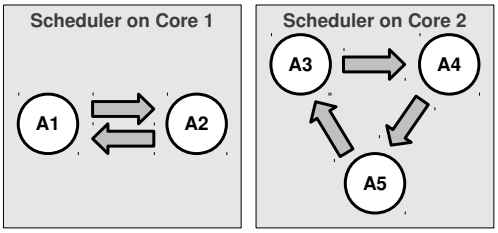


Fig. 5: Distributed multi-core scheduler using round-robin

The multi-core version of the data-driven / demand-driven strategy is realized in the same way as the round-robin strategy. The difference is that with our static mapping, the

predecessors and successors of a given actor can possibly be mapped to a different core than the one of this actor. This requires communications between the different threads to schedule all the actors. Figure 4 illustrates this: If A1 is blocked during its execution because the *data-fifo* called f2 is full then the scheduler has to add A3 to the list of schedulable actors. However A3 is managed by another scheduler so an inter-scheduler communication is needed.

In a multi-core context, we use a combined version of the round-robin and data-driven / demand-driven strategies to avoid starvation of our distributed algorithm. Indeed, contrary to the single-core version, data-driven / demand-driven strategy cannot guarantee all the time that each local schedulable list is not empty. The scheduler applies the data-driven / demand-driven policy until its schedulable list is empty and then the round-robin policy is used until the schedulable list contains at least one actor. The algorithm is presented in Fig. 6.

```
CombinedScheduling()
begin
  while true do
        if isEmpty(schedulable)
          then actor = getNext(RoundRobin);
          else actor = getNext(DataDemandDriven);
        fi;
        fire(actor);
        if ¬RoundRobin ∨ ||firing|| > 0
          then
                if isEmpty(actor.inputs)
                  then addPredecessors(actor);
                  else addSuccessors(actor);
                fi;
        fi;
  od;
end
```

Fig. 6: Combined scheduling algorithm

4.2. Lock-free inter-core communications

Lock-free communications between distributed schedulers are used to avoid the synchronization of threads. Indeed, the fine granularity of the actors makes actor scheduler a critical part of the execution so the smallest overhead can have disastrous consequences on the performance. Moreover informing a remote scheduler to add a schedulable actor to its schedulable list is essential otherwise a deadlock could occur during the execution. In fact, if this scheduling information is not communicated, a self-contained cycle can appear and cause a deadlock of the application. For example in Fig. 4, if the *data-fifos* f4-7 are full, the actors A3, A4 and A5 form a self-contained cycle that never stops until A2 consumes the tokens contained in f4 but this could never happen if the other scheduler loops on a self-contained cycle, too.

Another kind of FIFO channels called *scheduling-fifos* is used to communicate between schedulers without synchronization. Lamport proved that locks are not necessary in the case of single producer, single consumer FIFOs [7]. However it is important not to confuse the two types of FIFOs which work with the same mechanism but have two distinct uses: the *data-fifo* channels used to carry on the application stream and the *scheduling-fifo* channels used to share scheduling informations (in our case a set of next schedulable actors).

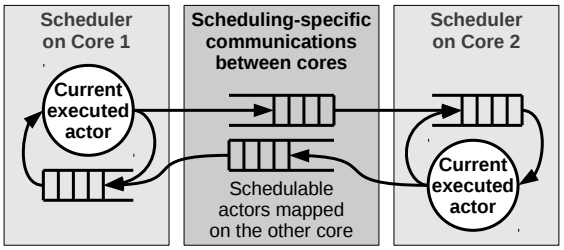
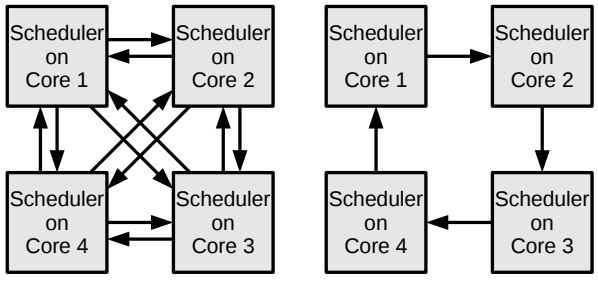


Fig. 7: Example of data-driven / demand-driven scheduling

Figure 7 shows the inter-core communications mechanism. When an actor execution is blocked, the scheduler adds the predecessor or the successors of the blocking *data-fifo* to its schedulable list. In some cases this actor is not executed by the current scheduler so this actor is sent to its associated scheduler by a *scheduling-fifo* channel.

We propose two kinds of communication network topology (Fig. 8): mesh and ring. The mesh topology uses a bidirectional communication channel between each couple of actors. The distributed schedulers communicate directly but the number of *scheduling-fifos* increases exponentially with the number of cores. The ring topology offers the possibility to use the distributed scheduler with a limited number of *scheduling-fifos*: on a N-core processors N *scheduling-fifos* are needed. However the communication could cross N-2 schedulers in the worst case before the targeted scheduler receives it. For example (see Fig. 8(b)), if the scheduler on core 1 wants to communicate with the other one mapped on core 4, the schedulers on cores 2 and 3 are used as intermediaries.



(a) Mesh (b) Ring

Fig. 8: Possible topologies of communications

5. RESULTS

In this section, we present several experimental results to demonstrate the efficiency of our multi-core scheduler on real-world video applications. We also compare our approach with another runtime included in the OpenDF framework on the same applications [6] [11].

5.1. Benchmarks

The Reconfigurable Video Coding (RVC) framework was created by MPEG to increase the reusability and portability of the code in video decoders [5]. The description language of RVC, called RVC-CAL and based on the Dataflow Process Networks model, was used to implement the applications used in our experiments.

We have implemented the round-robin and combined strategies in the C runtime library of the Open RVC-CAL Compiler (Orcc)[1]. These two scheduling strategies have been tested on dataflow descriptions of MPEG-4 Simple Profile and MPEG-4 Advanced Video Coding with different sized video sequences. We benchmarked these decoders on a Intel Xeon with four cores at 2.33GHz. During all the experiments, the *data-fifos* are bounded to 4096 elements and the *scheduling-fifos* to 200 elements.

The testing video sequences are: For MPEG-4 SP, hit001 (CIF) from ISO/IEC 14496-4:2004 and old_town_cross (720p) encoded at 6Mbps with the Xvid encoder from an YUV file available on [12] and for MPEG-4 AVC LS_SVA_D (QCIF) and HCBP2_HHI_A (CIF) available on [13].

The results for various configurations are presented in Table 1 for the MPEG-4 SP decoder and in Table 2 for the MPEG-4 AVC decoder. We benchmarked too the MPEG-4 SP decoder on the 720 sequence using the other runtime library included in OpenDF framework: We obtain 10.1 fps on one core and 15.6 fps on two cores i.e. the speedup is about 1.54.

Strategy		Core	CIF	720p	Speedup
Round-robin		1	144	15.6	1
		2	265	26.6	1.78
		4	494	51.4	3.36
Combined strategy		1	154	16.1	1
		2	288	27.3	1.75
	Ring	4	443	49.8	2.98
	Mesh	4	516	51.9	3.28

Table 1: Results of MPEG-4 SP for various configurations in frames per second

[1]Orcc is available at http://orcc.sf.net

Strategy		Core	QCIF	CIF	Speedup
Round-robin		1	28.4	7.1	1
		2	55.6	13.9	1.96
		4	90.8	21.2	3.05
Combined strategy		1	169	40.6	1
		2	294	71.4	1.74
	Ring	4	341	75.2	1.93
	Mesh	4	473	97.1	2.59

Table 2: Results of MPEG-4 AVC for various configurations in frames per second

5.2. Mapping validation using genetic algorithm

A genetic algorithm was developed to find efficient static mapping of actors on the processor cores and used during these experiments. Most of the time, dynamic dataflow programs can be easily partitioned on dual-core processor with manual methods thanks to the explicit parallelism of dataflow representations. However this is increasingly complex to do when the number of cores and actors grows.

For example, the dataflow description of MPEG-4 SP is composed of 42 actors and MPEG-4 AVC one is composed of 131 actors. More than one thousand possible mappings of a dataflow program into a multi-core processor is quickly reached.

5.3. Discussion

The data-driven / demand-driven strategy shows its efficiency with the MPEG-4 AVC decoder which contains many actors and many control flows. Moreover the data-driven / demand-driven strategy, and consequently the combined strategy, is slightly more efficient than the round-robin strategy even on small applications like MPEG-4 SP.

Our multi-core extension of these single-core scheduling strategies is validated by the high speedups we obtained compared to the maximal theoretical speedups.

Results show that the round-robin strategy is better on MPEG-4 Simple Profile than the data-driven / demand-driven strategy on four cores. Indeed, most of the time the round-robin executes a fireable actor because this application is described with few actors. When MPEG-4 SP is partitioned on multiple cores, more scheduling operations are required using the data-driven / demand-driven strategy, which leads to a slightly increased overhead.

Finally, the speedup obtained with the OpenDF framework is lower than the ones obtained with our two scheduling strategies with the same mapping of actors on the processor cores.

6. RELATED WORK

In [10] and [14], Parks and Haid et al. deal with implementation and scheduling of Kahn Process Networks. Contrary to Dataflow Process Networks, the context switches of process suspension and resumption cannot be avoided and it leads to an inevitable overhead. On the one hand, Haid et al. have chosen to use lightweight and stackless threads implementation to minimize this overhead. On the other hand, Parks presents a combined demand-driven and data-driven strategy. Unfortunately he gives no result we can use for comparison.

Aldinucci et al. presents a low-level programming framework based on lock-free queues dedicated to multiprocessors streaming applications in [15]. Like us, they try to use lock-free communication channels to avoid synchronization between threads. In their model, all channels with multiple writers or/and readers are built by assembling a set of single reader and single writer channels with an external thread which manages the data copies between these channels. This approach avoids cache invalidation but a lot of data transfers is needed to hide the overhead of switching between threads.

In [16], Boutellier et al. present a methodology to map and schedule actors on multiprocessors. They begin to transform the RVC-CAL network in a set of homogeneous synchronous dataflow graphs. Unfortunately these graphs cannot be generated when an actor execution depends of the input token value. Moreover, the small MPEG-4 SP decoder was their only test-case and the complexity of such static techniques increases exponentially according to the application size.

7. CONCLUSION

This paper proposes a new approach to efficiently schedule dynamic dataflow programs with a lock-free and distributed algorithm on multi-core architectures based on two presented single-core scheduling strategies. The results of the experiments shows that our multi-core scheduler scales up well to four cores.

In future work we will focus on stream communications between cores to improve the application speedup and we will extend our multi-core scheduling algorithm to dynamically map the actors on processor cores.

8. REFERENCES

[1] D. May, "OCCAM," *SIGPLAN Not.*, vol. 18, pp. 69–79, April 1983.

[2] P. S. Pacheco, *Parallel programming with MPI*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[3] M. Cole, *Algorithmic skeletons: structured management of parallel computation*, MIT Press, Cambridge, MA, USA, 1991.

[4] J.T. Buck and E.A. Lee, "Scheduling dynamic dataflow graphs with bounded memory using the token flow model," *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, vol. 1, pp. 429–432, 1993.

[5] I. Amer, C. Lucarz, G. Roquier, M. Mattavelli, M. Raulet, J.-F. Nezan, and O. Deforges, "Reconfigurable video coding on multicore," *Signal Processing Magazine, IEEE*, vol. 26, no. 6, pp. 113 –123, nov. 2009.

[6] C. von Platen, "Project report: CAL ARM Compiler," http://www.actors-project.eu/, Jan. 2010.

[7] L. Lamport, "Specifying Concurrent Program Modules," *ACM Trans. Program. Lang. Syst.*, vol. 5, pp. 190–222, April 1983.

[8] E. A. Lee and T. Parks, "Dataflow Process Networks," in *Proceedings of the IEEE*, 1995, pp. 773–799.

[9] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Information Processing '74: Proceedings of the IFIP Congress*, J. L. Rosenfeld, Ed., pp. 471–475. North-Holland, New York, NY, 1974.

[10] T. M. Parks, *Bounded Scheduling of Process Networks*, Ph.D. thesis, EECS Department, University of California, Berkeley, 1995.

[11] S. S. Bhattacharyya, G. Brebner, J. W. Janneck, J. Eker, C. von Platen, M. Mattavelli, and M. Raulet, "OpenDF: a dataflow toolset for reconfigurable hardware and multicore systems," *SIGARCH Comput. Archit. News*, vol. 36, pp. 29–35, June 2009.

[12] "YUV Sequences," http://media.xiph.org/video/derf/.

[13] "H264 Sequences," http://wftp3.itu.int/av-arch/jvt-site/.

[14] W. Haid, L. Schor, K. Huang, I. Bacivarov, and L. Thiele, "Efficient Execution of Kahn Process Networks on Multi-Processor Systems Using Protothreads and Windowed FIFOs," in *Proc. IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, Grenoble, France, 2009, pp. 35–44, IEEE.

[15] M. Aldinucci, M. Meneghin, and M. Torquati, "Efficient Smith-Waterman on Multi-core with FastFlow," in *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, Feb. 2010, pp. 195 –199.

[16] J. Boutellier, V. Martin Gomez, O. Silven, C. Lucarz, and M. Mattavelli, "Multiprocessor scheduling of dataflow models within the Reconfigurable Video Coding framework," in *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2009.