Fareed Jokhio | Andreas Dahlin | Johan Ersfolk | Johan Lilius

Analysis of an RVC-CAL MPEG-4 Simple Profile Decoder

Turku Centre *for* Computer Science

TUCS Technical Report
No 1018, September 2011

Analysis of an RVC-CAL MPEG-4 Simple Profile Decoder

Fareed Jokhio

Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5, 20520 Turku, Finland
fjokhio@abo.fi

Andreas Dahlin

Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5, 20520 Turku, Finland
andalin@abo.fi

Johan Ersfolk

Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5, 20520 Turku, Finland
jersfolk@abo.fi

Johan Lilius

Åbo Akademi University, Department of Information Technologies
Joukahaisenkatu 3-5, 20520 Turku, Finland
jolilius@abo.fi

TUCS Technical Report
No 1018, September 2011

Abstract

Profiling and instrumentation can be used to identify the inefficient areas of the code which may require optimization. In this technical report profiling and instrumentation results for RVC-CAL MPEG-4 simple profile decoder are discussed. The decoder is using dynamic scheduling for action invocation in its all networks. The IDCT part is optimized and dynamic scheduling of actions is replaced by static scheduling. Profiling and instrumentation results show that the number of operations performed by dynamically scheduled dataflow network are significantly reduced when part of the dataflow network is statically scheduled.

Keywords: MPEG-4 simple profile, CAL actor language, Reconfigurable Video Coding, RVC-CAL, profiling, instrumentation, dataflow modelling, dataflow programming, quasi-static scheduling, benchmarking, parallel processing

TUCS Laboratory
Embedded Systems Laboratory

1. Introduction

The number of users watching the video content on the internet is growing rapidly. Video consists of huge amount of data and it is therefore sent to the end user in compressed form. It is also efficient to store the video in compressed form on storage media such as hard disk or CD. In order to be able to watch the video content a video decoder is required. The video decoder requires a significant amount of processing power therefore multi-core processor platforms can be suitable for such applications. A decoder implemented using conventional programming languages such as C/C++ cannot easily be distributed on multi-core platforms. For such platforms a decoder expressed as a dataflow network is more suitable. The video decoder represented as dataflow network has nodes and arcs. The nodes in a dataflow network represent computational entities and the arcs show the flow of data in the network. The dataflow network is a directed graph and can have any number of inputs and outputs at a node.

In the decoder there are several different operations which need to be performed and there are interdependencies among those operations. If the number of operations or processes is larger than the number of processing units available, then a scheduler is required in order to arrange those operations which are mapped on the same processing unit into an appropriate sequence. The scheduling can be performed at compile time (static scheduling) or at run time (dynamic scheduling). In static scheduling the order of executing different processes is predefined and it is not needed to check whether those processes are eligible for execution or not. In the dynamic scheduling the order of executing different processes is decided at run time so it is essential to check that whether they are eligible or not. If the number of processes which needs to be executed is very large and they need to be executed several number of times in a particular program, then significant amount of processor time will be consumed in checking for the eligibility of the processes. The extra overhead, caused by checks, will slow down the overall performance of the application. In a video decoder the operations which needs to be performed depend on the content of data hence it is not possible to get static scheduling. But it is quite possible to find out certain set of processes which always execute in a particular order or sequence. This kind of scheduling is termed as quasi-static scheduling and it can be helpful in reducing the overheads.

In the quasi-static scheduling approach compile time analysis are applied to improve runtime decisions. A quasi-static schedule has both static scheduling sequences and runtime decisions as shown in figure 1. Based on a run time decision one of those static sequences is selected for execution. In quasi-static scheduling data dependent scheduling decisions are made at runtime and static decisions are made at compile time.

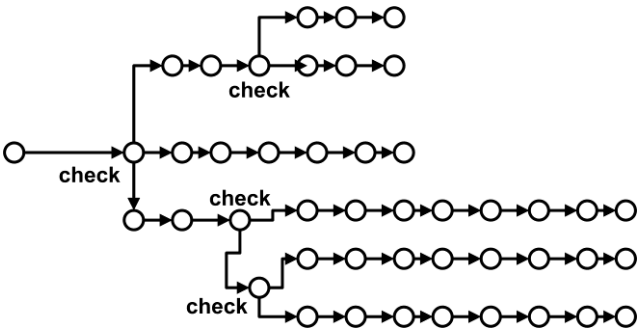

Figure 1. Graphical Representation of a Quasi Static Schedule¨

In order to know the overheads in the decoder, profiling and instrumentation can be used and inefficient areas having more overhead can be identified. The main contribution of this work is:

- Perform the profiling and instrumentation for the dynamically scheduled decoder and find the overheads in its different parts.
- Perform the profiling and instrumentation for the same decoder having static schedule on its IDCT network and find the reduction in the overheads.

2. Background and Related Work

A new video coding standard named Reconfigurable Video Coding (RVC) [8] is developed by the MPEG group. This RVC framework was standardized by ISO/IEC in 2009. The main objective of this standard is to provide an open framework which is capable of specifying and reconfiguring video codecs by connecting different video coding tools or functional units. In the RVC framework there are three languages used: RVC-CAL [1, 2], FNL [10] and RVC-BSDL [11]. RVC-CAL is a subset of a dataflow programming language named Caltrop Actor language (CAL) [9] which is used for modelling Functional units (FUs). The FNL is the Functional Units Network Language which is used for describing connections of the RVC-CAL Functional Units. The RVC-BSDL (Bit stream Syntax Description Language) is used for describing the syntax of the bit stream.

A CAL program consists of actors. An actor performs some task or computation, it can accept input as well as can produce output, it can have internal states and an actor is completely independent from other actors. The actor state is internal to the actor and cannot be observed by other actors, which means that actors are independent units, which can interact with each other by exchanging data named as token through channels [4]. An actor in network fires its actions based on status of input tokens and guard conditions.

The execution of a CAL program is asynchronous, actors can be fired in any order and any actor in the network can be fired if one of its actions is eligible. An action is eligible if there are sufficient input tokens, there is space on the output buffer and the guard condition is satisfied[4]. A CAL application often contains a large number of actors. If more than one actor is executed on a single processing unit then an inter-actor scheduler is required. The CAL actors are dynamic and it is not possible to schedule them statically, because the current state of the actor and its input values determine which actions can be fired. The runtime scheduler needed for those actors causes extra overheads or checks before firing any action.

The model checkers are used to verify the requirements and design for a system. In [12, 13, 14, and 15] model checkers are used to get the schedule for dataflow network. In [13] Boutellier worked with scheduling of a RVC-CAL network and proposed a quasi-static schedule. He did not use a model checker; the proposed schedules were derived by hand. Thereby generating schedules for different decoders will be hard and time consuming.

In [3] Ersfolk proposes a model checking approach, which uses prior knowledge to identify dynamic sub-networks with the model checker. With this approach dynamic scheduling is replaced with quasi-static scheduling. In this paper a new schedule is obtained based on that approach, for the IDCT part of the MPEG-4 Simple profile decoder. The obtained schedule is a single static schedule for the entire IDCT network.

2.1. Profiling and Instrumentation

To get the timing information and CPU load costs for an application, instrumentation and profiling can be used. It allows gathering of timing information for all functions in a program. The profiler lets us view performance of programs and thus it indicates inefficient areas of the code which may require more optimization.

Program profiling from software engineering point of view is used for optimizing a program; the tool used for performing the profiling is the profiler which is used for performance analysis. The profiler is used to get the behaviour of program from program starting to program ending. With profiling following measures can be performed:

1. Memory usage
2. Instructions usage
3. Function calls frequency
4. Function calls duration

The profile of a program obtained by profiler depends on the following parameters:

1. Application source code
2. Compiler settings
3. Software and hardware platform it runs on

The code instrumentation technique is used for data collection. In order to check how many number of times a particular section or part of a function is executed, code instrumentation can be used. In this report the code instrumentation is used to count the number of checks which fails or number of misses is counted with code instrumentation.

3. RVC-CAL MPEG-4 Simple Profile Decoder

In the RVC-CAL MPEG-4 Simple Profile Decoder the main functional blocks are the bit stream parser, the reconstruction block, the 2D IDCT, the frame buffer, and motion compensation [17]. Top level view of the MPEG-4 simple profile decoder is shown in figure 2. Input to the decoder is the compressed bit stream and output is the series of pictures or a video.

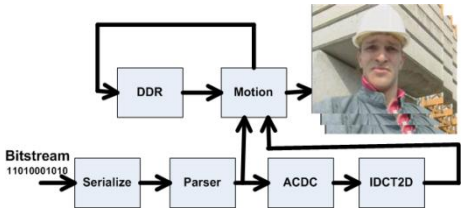

Figure 2. Block Diagram of the RVC-CAL MPEG-4 Simple Profile Decoder

The decoder consists of several networks and sub networks of actors. Inside an actor there are number of actions. The decoder consists of nine (9) sub-networks and 42 actor instantiations. A more detailed representation of the decoder is shown in figure 3.

The dataflow oriented MPEG-4 Simple profile decoder is scheduled dynamically. The original project with dynamic schedule is written in the RVC-CAL [1, 2] language. The CAL2C [5] code generator is used to generate C code from the RVC-CAL program so that it can be compiled and executed on most processors [6] including embedded processors. The dynamic schedule for the entire IDCT network is converted into a static schedule using the approach discussed in [3], which is used to get the quasi-static schedules by using model checkers. For the IDCT network, instead of getting quasi-static schedules, it is even possible to obtain a single static schedule.

The static schedule is implemented on the network level and it is independent of the state machine of the individual actors. It is possible to fire a set of actions of one actor and then fire a couple of other actions within the same network and again come back to fire actions of the same actor within the network. Hence if actions become ready to be fired then actors are allowed to fire actions and do not need not to wait for the complete cycle.

In this case study the static schedule is applied on only IDCT network of the MPEG-4 decoder. More detailed description about this decoder is available in [17]. All other networks and actors are dynamically scheduled and have their own state machine. The entire IDCT network is treated as single scheduling unit and is scheduled in round robin fashion with other actors.

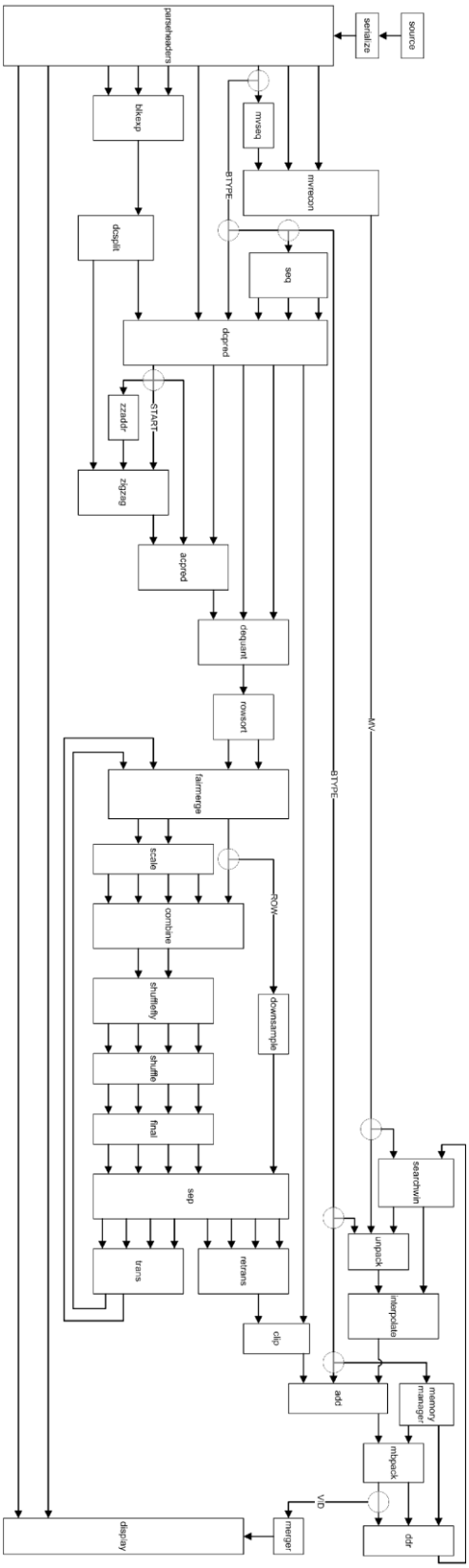

Figure 3. Actors in the RVC-CAL MPEG-4 Simple Profile Decoder

4. Experimental Setup

In order to get instrumentation and profiling results four video sequences were chosen, three *Foreman video sequences* consist of 300 frames. One Foreman video sequence only have intra coded frames while others have intra as well as progressive frame sequence format. The *Football video sequence* has 360 frames. In Foreman QCIF, Foreman CIF and Football QCIF video sequences 8% of all frames are of type I and 92 percent are of type P. The resolution for all sequences is 176x144 pixels, except one of the football sequences which has CIF (352x288 pixels) resolution. Table 1 shows resolution, number of frames and frames types for all test video sequences.

Video Sequence	Resolution	No. of Frames	Frame Types
Foreman QCIF	176x144	300	I 8% P 92%
Foreman QCIF Intra	176x144	300	I 100%
Foreman CIF	352x288	300	I 8% P 92%
Football QCIF	176x144	360	I 8% P 92%

Table 1. Test Video Sequences

In order to see the improvement in terms of frames per second the execution time analysis was performed on a desktop computer equipped with an Intel Core 2 Duo CPU running at 2.66 GHz, 2 GB of RAM and running the 32-bit version of Windows 7 Professional. Both instrumentation and profiling results were obtained using the same machine. In this case study the instrumentation and profiling results are obtained using Microsoft Visual Studio 2010.

In order to perform profiling in visual studio, the following steps can be followed:

1. Click on the Analyze menu
2. Select the Launch Performance Wizard
3. Then the performance wizard will be opened having 3 pages. On the first page you have to select the Instrumentation option. This option is used to measure function call counts and timing.
4. Click next and on second page select the project which needs to be instrumented.
5. Click next
6. On the third page you have to uncheck the Launch profiling after the wizard finishes option. This is essential to uncheck this option because the decoder programs needs the file name of compressed video sequence as an argument, which must be provided.
7. Click Finish. Now the Performance Explorer will be opened.
8. Right click on the project name under the Targets option in the performance explorer and select properties option.
9. The property dialog box will be shown having the following four options

    - Launch
    - Tier Interactions
    - Instrumentation
    - Advanced

10. Select the launch option
11. Check the Override project setting option and provide the compressed video sequence name in the Arguments text box.
12. Click on Instrumentation option on the right side of the property dialog box
13. Uncheck the option Exclude small functions from instrumentation option.
14. click on the ok button
15. The profiler can now be executed to obtain the results.

5. Results

In this section gain in terms of speedup for decoding process is discussed along with Instrumentation and Profiling results.

5.1. Performance Gain

The number of frames per second for different video sequences, before and after optimization of scheduling, is given in table 2. We observe that 13% gain can be obtained by scheduling the IDCT part with static schedule.

Video Sequence	Dynamic	Static	Gain
Foreman QCIF	81 fps	93 fps	13%
Foreman QCIF intra	84 fps	96.5 fps	13%
Foreman CIF	22 fps	25 fps	12%
Football QCIF	67 fps	77 fps	13%

Table 2. Frames per Second for Dynamic and Quasi-static Schedules

After applying the static schedule on IDCT network there is significant gain in execution time for IDCT part, it is also observed that there was slight improvement in other networks as a side effect. In the dynamic schedule the actors are invoked in round robin fashion and if an actor is not ready to be fired, and then it has to wait until its next turn. With the static scheduler unnecessary waiting time will be reduced which results in faster decoding operation.

Applying static scheduling does not degrade the performance of other networks. Either it will either improve their performance or will have no effect.

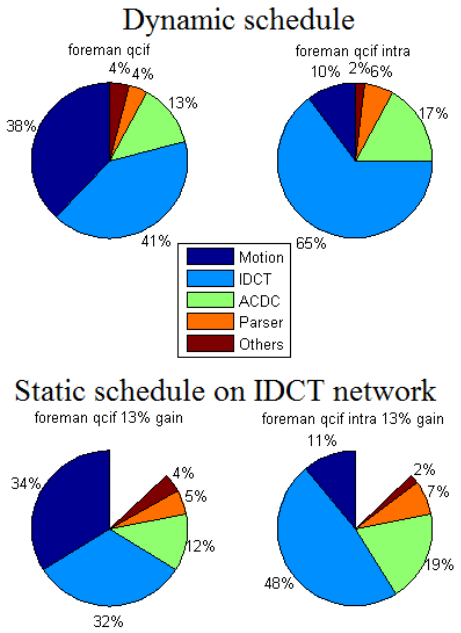

Figure 4. Foreman QCIF and Foreman QCIF Intra Video Sequences Gain

In figure 4, execution time in percentage for Foreman QCIF and Foreman QCIF Intra video sequences is indicated for both static and dynamic schedules. For Foreman QCIF video sequence both motion and IDCT are time consuming tasks. There is significant gain in the IDCT part after applying the static schedule and a small gain in the Motion part as a side effect.

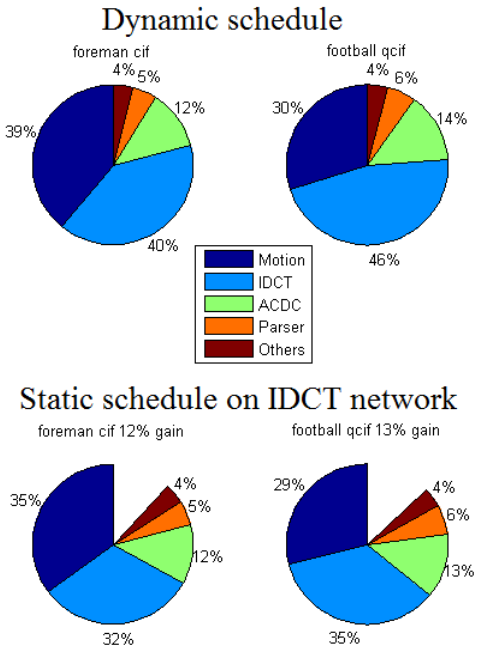

Figure 5. Foreman CIF and Football QCIF Video Sequences Gain

In figure 5, execution time in percentage for Foreman CIF and Football QCIF video sequences is indicated for both static and dynamic schedules. There is 12% gain for the Foreman CIF video sequence and 13% gain for football video sequence.

It is worth to mention here that all results are obtained using Windows 7 operating system. The operating system itself executes a number of other processes while performing decoding operation; hence there will be nondeterministic behaviour in the results.

5.2. Instrumentation Results

For MPEG-4 decoder with dynamic scheduling it was observed that there are millions of firings of actions, hence for every firing there is one check or several checks. It means that the decoder is using some part of time in checking those guard conditions or overheads.

By using static scheduling as proposed in [3] this number of checks can be reduced and significant speedup is possible in decoding operation. In experiments with different video sequences it was observed that more than 40% of those checks produce false results hence no firing of actions. The table 3 summarizes the number of checks for both static and dynamic scheduled MPEG-4 Decoder.

Video Sequence	Dynamic	Static	Reduction
Foreman QCIF	$300x10^6$	$242x10^6$	19.38%
Foreman QCIF Intra	$402x10^6$	$238x10^6$	40.60%
Foreman CIF	$1142x10^6$	$938x10^6$	17.87%
Football QCIF	$419x10^6$	$322x10^6$	23.19%

Table 3. Total Number of Checks for Dynamic and Static Schedules

In table 3 it can be observed that the number of checks is reduced by applying the static schedule on the IDCT network. For different actions the time consumed in guard conditions checking is different. If any guard condition is checked then there can be multiples checks for it. It can fail at the very first check means there is no need to check further conditions or it may fail at the very last check. Here in this report the term "miss" indicates that the guard condition was checked and it produced false result and no action was fired. If all conditions become true for action's guard then the action is fired and here we use the term "Hit".

The results for hits and misses are shown in table 4. It can be observed that for all video sequences there is reduction in number of hits and misses after applying the static schedule. The reason for this reduction is that we have only a single point of check for entire IDCT network.

In order to calculate number of hits and misses the original code was instrumented by inserting extra code. Figure 6 shows the code listing of serialize actor. The bold code is the instrumented code. Serialize actor has two actions reload and shift. In the C language code there is one function for each actor. Due to space limitation body of the functions is not indicated. The profiler gives the number of invocations for every actor, hence in order to calculate number of hits there is no need to insert or instrument code. The number of misses cannot be viewed from the profiler results for the original code. Hence two extra functions are created having no code in body. These functions are called whenever a miss occurs. After inserting these two extra functions bodies and their calls at appropriate locations, it is possible to view the number of missed in the profiler results.

The code instrumentation is done for all actors, hence there is one extra function for every action and it is called whenever a misses occurs for that action. The sum of the misses of actions inside an actor gives the total number of misses for that particular actor.

```
// /Xilinx_top/decoder/serialize
void serialize_reload_not(){} //this is instrumented code
void serialize_reload() { … }
static i32 isSchedulable_reload() { … }
void serialize_shift_not() {} //this is instrumented code
void serialize_shift() { … }
static i32 isSchedulable_shift() { … }

// Action scheduler
void decoder_serialize_scheduler(struct schedinfo_s *si) {
  int i = 0;
  while (1) {
    if (fifo_u8_has_tokens(decoder_serialize_in8, 1) &&
        isSchedulable_reload()) {
      serialize_reload();
      i++;
    } else if (isSchedulable_shift()) {
      int ports = 0;
       serialize_reload_not();  //this is instrumented code
      if (!fifo_i32_has_room(decoder_serialize_out, 1)) {
        ports |= 0x01;
      }
      if (ports != 0) {
        si->num_firings = i;
        si->reason = full;
        si->ports = ports;
        serialize_shift_not(); //this is instrumented code
        break;
      }
      serialize_shift();
      i++;
    } else {
      si->num_firings = i;
      si->reason = starved;
      si->ports = 0x01;
       serialize_reload_not();  //this is instrumented code
       serialize_shift_not();    //this is instrumented code
      break;
    }
  }
}
```

Figure 6. Code Listing for Serialize Actor

Video Sequence	Dynamic Schedule		Static Schedule for IDCT	
	Hits	Misses	Hits	Misses
Foreman QCIF	$151x10^6$	$148x10^6$	$106x10^6$	$135x10^6$
Foreman QCIF intra	$233x10^6$	$169x10^6$	$110x10^6$	$128x10^6$
Foreman CIF	$569x10^6$	$572x10^6$	$412x10^6$	$525x10^6$
Football QCIF	$217x10^6$	$202x10^6$	$143x10^6$	$179x10^6$

Table 4. Summary of Hits and Misses for Dynamic and Static Schedules

In table 5 the number of hits and misses is indicated for both original and static schedules for Foreman QCIF video sequence. It can be noticed that there is large difference in the number of hits and misses for the dynamic schedule and static schedule.

Sequence	Dynamic Schedule		Static Schedule for IDCT	
	No. of Hits	No. of Misses	No. of Hits	No. of Misses
Motion	62 577 963	98 854 178	62 570 890	98 384 798
IDCT	44 881 058	11 988 769	64 941	46 586
ACDC	23 923 793	24 233 886	23 923 570	23 797 153
Parser	6 978 669	11 162 185	6 978 439	10 782 984
Others	13 191 622	2 745 103	13 189 062	2 552 495
Total	151 553 105	148 984 121	106 726 902	135 564 016

Table 5. Number of Hits and Misses for Foreman QCIF Video Sequences

In table 6 the number of hits and misses is indicated for both original and static schedule for Foreman QCIF intra video sequence. In this video sequence all frames are I type.

Sequence	Dynamic Schedule		Static Schedule for IDCT	
	No. of Hits	No. of Misses	No. of Hits	No. of Misses
Motion	24 494 875	36 424 864	24 495 983	34 241 049
IDCT	123 076 178	33 306 888	178 119	71 221
ACDC	58 979 069	61 499 394	58 980 005	59 178 581
Parser	15 464 236	28 881 422	15 464 092	27 072 505
Others	11 309 009	8 960 760	11 320 955	7 992 852
Total	233 323 367	169 073 328	110 439 154	128 556 208

Table 6. Number of Hits and Misses for Foreman QCIF Intra Video Sequences

In table 7 the number of hits and misses is indicated for both original and static schedule for Foreman CIF video sequence. Since the frame size in CIF video sequence is bigger

than the QCIF video sequence hence more time is required in decoding the video frames. In other words more actions need to be fired, hence more checks before firing the actions. The number of checks will be different for different video sequences. Larger frame size will result in more checks, but even with same size of frames this number will be different because it will totally depend on the types of operations which will be performed during the decoding.

Sequence	**Dynamic Schedule**		**Static Schedule for IDCT**	
	No. of Hits	No. of Misses	No. of Hits	No. of Misses
Motion	250 724 857	395 531 580	250 727 476	393 982 757
IDCT	157 168 871	42 367 866	227 416	187 169
ACDC	84 347 764	85 539 738	84 348 375	84 037 814
Parser	25 581 425	39 921 813	25 581 668	38 634 437
Others	51 462 690	9 626 921	51 459 405	8 971 186
Total	569 285 607	572 987 918	412 344 340	525 813 363

Table 7. Number of Hits and Misses for Foreman CIF Video Sequences

In table 8 the number of hits and misses is indicated for both original and static schedule for football QCIF video sequence. This video sequence has more misses in the Motion compensation.

Sequence	**Dynamic Schedule**		**Static Schedule for IDCT**	
	No. of Hits	No. of Misses	No. of Hits	No. of Misses
Motion	74,667,161	118,614,059	74,671,073	117,526,842
IDCT	74,117,194	19,944,196	107,261	57,715
ACDC	38,003,615	38,750,133	38,006,407	37,743,143
Parser	11,279,761	18,626,702	11,280,228	17,764,073
Others	19,119,406	6,425,015	19,116,285	5,972,302
Total	217,187,137	202,360,105	143,181,254	179,064,075

Table 8. Number of hits and misses for football QCIF video sequences

5.3. Profiling Results

Table 9 indicates the decoding and overheads time in percentage for Foreman QCIF video sequence. It can be noticed that the overhead time in IDCT part was 27% in the dynamic schedule and after applying the static schedule it is only 2.95%. The table indicates that decoding time for motion was 67.7% in the dynamic schedule and it is 68.21% in the static schedule. In reality the decoding time is not changed it is constant in both schedules. Since overheads time is reduced hence the total time is also reduced so the scale of time is not same.

Sequence	Dynamic Schedule		Static Schedule for IDCT	
	Decoding time	Overhead time	Decoding time	Overhead time
Motion	67.69%	32.31%	68.21%	31.79%
IDCT	72.89%	27.11%	97.04%	2.96%
ACDC	52.74%	47.26%	49.49%	50.51%
Parser	47.94%	52.06%	40.85%	59.15%
Others	63.02%	36.98%	61.25%	38.75%
Total	66.90%	33.10%	74.34%	25.66%

Table 9. Decoding and Overhead Timings for Foreman QCIF Video Sequence

In table 9 the total decoding time is 66.9% in the dynamic schedule and 33.1% time is the overheads time. There is no reduction or change in the decoding time. The only change is in the overheads time and it is significantly reduced hence total time becomes less. Hence now the decoding time is 74.3% and still we have 25.66% overheads time. It is possible to apply the quasi-static schedule on other parts of decoder to get more gain and reduce the overheads time as much as possible.

The table 10 indicates the decoding and overheads time in percentage for Foreman QCIF intra video sequence. For Intra frames motion compensation is not required and even there is no need of any reference frame for decoding the current frame, but the decoder does not know in advance that which type of frame will be coming after the current frame, hence it stores the current frame as a reference frame.

Sequence	Dynamic Schedule		Static Schedule for IDCT	
	Decoding time	Overhead time	Decoding time	Overhead time
Motion	70.65%	29.35%	73.27%	26.73%
IDCT	74.31%	25.69%	97.21%	2.79%
ACDC	52.72%	47.28%	53.47%	46.53%
Parser	49.118%	50.89%	46.84%	53.16%
Others	59.14%	40.86%	55.69%	44.31%
Total	68.46%	31.54%	79.44%	20.56%

Table 10. Decoding and Overhead Timings for Foreman QCIF Intra Video Sequence

Table 10 indicates that there are 20.5% overheads in the static schedule. The 2.79% overheads in the IDCT part are due to some checks which are very essential to fire any action. The context switch time in firing the actions is also included in this overheads time. Current implementation of the static schedule does not inline the actions code

hence it introduces a little overhead. Table 11 indicates the decoding and overheads time in percentage for Foreman CIF video sequence.

Sequence	**Dynamic Schedule**		**Static Schedule for IDCT**	
	Decoding time	Overhead time	Decoding time	Overhead time
Motion	67.40%	32.60%	66.60%	33.40%
IDCT	74.19%	25.81%	97.10%	2.90%
ACDC	52.04%	47.96%	51.56%	48.44%
Parser	47.84%	52.16%	49.80%	50.20%
Others	62.66%	37.34%	63.95%	36.05%
Total	67.11%	32.89%	73.59%	26.41%

Table 11. Decoding and Overhead Timings for Foreman CIF Video Sequence

The decoding process on desktop machines is nondeterministic because other threads are also running hence it will not give 100% same timing results on the same machine for same video sequence. During experiments it was noted that there was always 2% or even more variation in the execution time. The above results were taken during different executions hence they may not indicate 100% same type of values but however they indicate same information that static schedule always performs better for almost all video sequences.

The table 12 indicates the decoding and overheads time in percentage for football QCIF video sequence. The overheads time after applying static schedule is 3.1%.

Sequence	**Dynamic Schedule**		**Static Schedule for IDCT**	
	Decoding time	Overhead time	Decoding time	Overhead time
Motion	68.79%	31.21%	68.117%	31.89%
IDCT	73.77%	26.23%	96.91%	3.09%
ACDC	52.93%	47.07%	51.77%	48.23%
Parser	48.42%	51.58%	48.93%	51.07%
Others	62.53%	37.47%	62.68%	37.32%
Total	67.39%	32.61%	75.44%	24.56%

Table 12. Decoding and Overhead Timings for Football QCIF Video Sequence

It can be noted in all above results that the overheads time after applying static schedule is almost 3% for all video sequences in the IDCT network. It means that 97% of the time is consumed in doing the real decoding in this network.

6. Conclusion and Future Work

In this case study the profiling and instrumentation results are obtained for the RVC-CAL Simple Profile Decoder. The original decoder has dynamic scheduling.

Instrumentation results show the number of checks which are placed before action invocation. These checks are extra overheads in the decoding process. The profiling results show the actual decoding time and overheads time. It was observed that more than 33% of the total decoding time is utilized in the overheads. The static schedule is obtained for the IDCT part by using the approach discussed in [3]. Instrumentation results are obtained for the decoder having static scheduling in IDCT network, the results shows that for different video sequences there was significant reduction of the checks. The profiling results show 13% gain is obtained in overall decoding time with better scheduling.

It was further measured that the IDCT part with dynamic scheduling has more than 27% overheads. With static scheduling in the IDCT part the overheads time is less than 3%, which is significant gain.

In the future we plan to obtain quasi-static schedules for the other networks for MPEG-4 Simple Profile decoder and thereby obtain increased performance. The approach for finding quasi-static schedules should be further validated by profiling and instrumenting other dataflow applications as well.

References

[1] ISO/IEC 23001-4:2009, "Information technology - MPEG systems technologies - Part 4: Codec configuration representation," 2009

[2] Marco Mattavelli, Ihab Amer, and Michael Raulet, "The reconfigurable video coding standard", IEEE Signal Processing Magazine, vol. 27, no. 3, pp 157-167, 2010.

[3] Johan Ersfolk, Ghislain Roquier, Fareed Jokhio, Johan Lilius, Marco Mattavelli, “Scheduling of dynamic dataflow programs with model checking,” “IEEE Workshop on Signal Processing Systems SiPS 2011”, October 4-7 2011, Beirut, Lebanon

[4] Nicolas Siret, Ismail Sabry, Jean Francois Nezan and Mickael Raulet, "A codesign synthesis from an MPEG-4 decoder dataflow description", "Circuits and Systems(ISCAS), Proceedings of 2010 IEEE International Symposium on France "

[5] G. Roquier, M. Wipliez, M. Raulet, J.W. Janneck, I.D. Miller, and D.B. Parlour, “Automatic software synthesis of dataflow program: An MPEG-4 simple profile decoder case study,” in Signal Processing Systems (SiPS). IEEE Workshop on, 2008, pp. 281–286.

[6] I. Amer, C. Lucarz, G. Roquier, M. Mattavelli, M. Raulet, J.-F. Nezan, and O Deforges, “Reconfigurable video coding on multicore,” in Signal Processing Magazine, IEEE, 2009, pp. 113–123.

[7] M. B. Taylor, J. Kim, J. Miller, D.Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, Jae-Wook Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, .The Raw microprocessor: a computational fabric for software circuits and general-purpose programs,. IEEE Micro, vol. 22, pp. 25.35, March 2002.

[8] Lucarz, C., Mattavelli, M., Thomas-Kerr, J., & Janneck, J. W. (2007). Reconfigurable media coding: A new specification model for multimedia coders. In IEEE workshop on signal processing systems (pp. 481–486). Shanghai, China.

[9] Eker, J., & Janneck, J. W. (2003). CAL language report. UC Berkeley, Tech. Rep. UCB/ERL M03/48.

[10] ISO/IECFDIS23001-4 (2009).MPEGsystems technologies—Part 4: Codec configuration representation.

[11] International Standard ISO/IEC FDIS 23001-5: MPEG systems technologies—Part 5: Bitstream Syntax Description Language (BSDL).

[12] Marc Geilen, Twan Basten, and Sander Stuijk, “Minimising buffer requirements of synchronous dataflow graphs with model checking,” in DAC ’05, 2005.

[13] Zonghua Gu et al., “Static scheduling and software synthesis for dataflow graphs with symbolic modelchecking,” in RTSS ’07, 2007.

[14] Nan Guan et al., “Improving scalability of modelchecking for minimizing buffer requirements of synchronous dataflow graphs,” in ASP-DAC ’09, 2009.

[15] Weichen Liu et al., “An efficient technique for analysis of minimal buffer requirements of synchronous dataflow graphs with model checking,” in CODES+ISSS ’09, 2009.

[16] Jani Boutellier, Christophe Lucarz, S´ebastien Lafond, Victor Gomez, and Marco Mattavelli, “Quasi-static scheduling of cal actor networks for reconfigurable video coding,” Journal of Signal Processing Systems, 2009.

[17] S. S. Bhattacharyya, J. Eker, J. W. Janneck, C. Lucarz, M. Mattavelli, and M. Raulet, “Overview of the MPEG reconfigurable video coding framework,” Journal of Signal Processing Systems, 2009, DOI:10.1007/s11265-009-0399-3.

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Information Technologies

Turku School of Economics
- Institute of Information Systems Sciences

ISBN 978-952-12-2636-6
ISSN 1239-1891