International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-4 Number-3 Issue-16 September-2014

Analysis for Parallel Execution without Performing Hardware/Software Co-simulation

Muhammad Rashid

Abstract

Hardware/software co-simulation improves the performance of embedded applications by executing the applications on a virtual platform before the actual hardware is available in silicon. However, the virtual platform of the target architecture is often not available during early stages of the embedded design flow. Consequently, analysis for parallel execution without performing hardware/software co-simulation is required. This article presents an analysis methodology for parallel execution of video encoding applications targeting heterogeneous reconfigurable architectures without performing HW/SW co-simulation. We formulate the application performance on the target architecture with an equation. The equation shows the overhead factors that reduces the speedup of parallel execution. H264 video encoding application is taken as a case study.

Keywords

Parallel execution, application analysis, heterogeneous reconfigurable architectures, video encoding.

1. Introduction

Embedded systems are becoming more and more complex. It includes complexity of application as well as the architecture heterogeneity [1], [2], [3]. However, time-to-market is constantly decreasing. The importance of being able to keep pace with both trends is posing serious challenges. To reduce this gap, hardware/software co-simulation has been around for more than two decades and has shown a lot of potential for embedded systems design. The objective of hardware/software co-simulation is to find an optimized solution such that all the design constraints are satisfied [4]. It generally requires heterogeneous platforms combining general purpose processor with reconfigurable hardware such as FPGA (Field Programmable Gate Arrays).

Manuscript received July 26, 2014.
Muhammad Rashid, Department of Computer Engineering, Umm Al-Qura University, Makkah, Saudi Arabia.

The performance is improved by partitioning (mapping) an embedded application among software running on a microprocessor and reconfigurable hardware for parallel execution. Performance improvement by parallel execution on heterogeneous reconfigurable platforms depends upon three factors:

- The optimal mapping of application on the target architecture. The performance gains in these heterogeneous multicore architectures depend on effective application parallelization across multiple cores [5].
- Exploitation of potential parallelism [6]. To get most benefit of such a heterogeneous platform, it is essential to exploit the parallelism of the application [7].
- Data exchange registers between different processing cores of the target architecture [8].

Hardware/software co-simulation requires either the actual hardware platform or virtual prototype of the target platform which in turns require considerable time and efforts. The availability of such tools, such as virtual prototype of the target architecture, in early stages of the design flow is a real bottleneck. Consequently, there is no generally accepted methodology to separate applications onto hardware and software execution [9]. The separation of application into hardware and software parts is called partitioning [5] – [9].

The partitioning problem in heterogeneous systems is more critical due to the exponential growth of VLSI (Very Large Scale Integration) technology. At the same time, the continuous growing effort of developing multimedia standards with higher compression efficiency and more functionality have resulted in application complexity [3].

This article presents a performance analysis methodology for video encoding applications to estimate the expected performance of parallel execution on heterogeneous reconfigurable architectures without performing HW/SW co-simulation. The important steps of proposed methodology in this article are: separation of source code, parallelization at macro-block level, dependency

analysis of macro-blocks, and identification of slow down factors.

Separation of source code divides the original code into two types: frame data processing and Macro-block (MB) processing. Then, the MB processing part is parallelized between different processing elements of the target heterogeneous reconfigurable architecture. However, the maximum parallelism is not always achieved due to dependency among macro-blocks.

Therefore, dependency analysis is performed to formulate the sum of time duration before and after maximum parallelism. Finally, slow down factors are identified. To the best of our knowledge, no such methodology exists that can analyze the video encoding applications for heterogeneous reconfigurable architectures without performing HW/SW co-simulation.

H.264 video encoding application [10] serves as a case study and the target platform in this article is the Molen architecture [11] (a heterogeneous reconfigurable platform). H.264 is commonly used due to its high-quality coding characteristics of video contents at very low bit-rates. Molen architecture supports integrated hardware-software co-simulation starting from profiling and partitioning to synthesis and compilation.

The remainder of this article is organized into multiple sections: Section 2 provides the essential background information. Section 3 describes parallelization opportunities for video encoding applications. Section 4 presents the four steps of proposed performance analysis methodology: separation of source code, parallelization at macro-block level, dependency analysis of macro-blocks, and identification of slow down factors, described in Sections 5, 6, 7 and 8 respectively. Finally, Section 9 provides conclusion.

2. Essential Background Knowledge

Before discussing the proposed performance analysis methodology, we briefly summarize the Molen architecture and the H.264 video encoding application. Our methodology is general purpose and is not specific to a particular application or target architecture. The H.264 is being selected due to its wider acceptance as a video encoding application. Molen architecture is chosen because it represents the characteristics of a typical heterogeneous reconfigurable platform used in embedded applications.

Block diagram of Molen Architecture:
Molen architecture is used to speed up the applications execution by implementing its most critical functions as hardware accelerators, referred to as Custom Computing Units (CCUs). The Molen machine organization is shown in Figure 1.

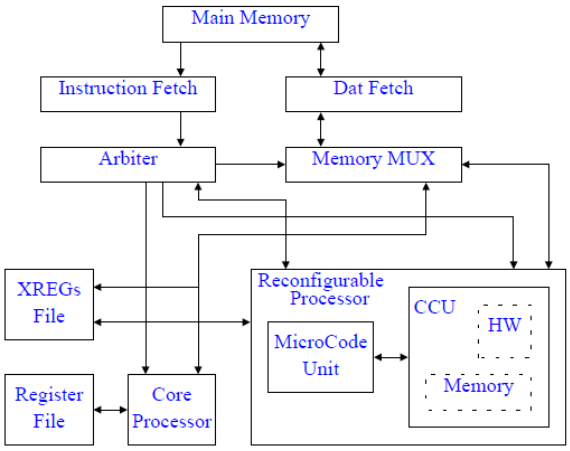

Figure 1: The Molen Machine Organization

The main parts are the general purpose processor (GPP), the reconfigurable co-processor (RP) and the Arbiter. The GPPs Instruction Set Architecture (ISA) is extended to control the hardware accelerators. The Arbiter fetches and decodes the application instructions from the main memory. It checks whether it belongs to the standard or to the extended ISA and arbitrates them to the corresponding processor. Exchange registers (XREG) are used for the transfer of data between GPP and the RP.

Block diagram of H.264 Video Application:
The block diagram of H.264 video encoding application (standard) is shown in Figure 2. The input frame (shown as Fn in Figure 2) is processed in units of macro- block. A macro- block corresponds to 16x16 pixels in the original image. Each macro-block of the target image is encoded in either intra or inter prediction mode. In each prediction mode, a macro-block P is constructed.

In Intra prediction mode, macro-block P is constructed from samples in the current frame that have previously been encoded, decoded and reconstructed. In Inter prediction mode, macro-block

P is constructed by motion-estimated (ME) and motion-compensated (MC) prediction from one or more reference frame(s). Once the macro-block P is constructed, it is subtracted from the current macro-block to produce a difference macro-block Dn.

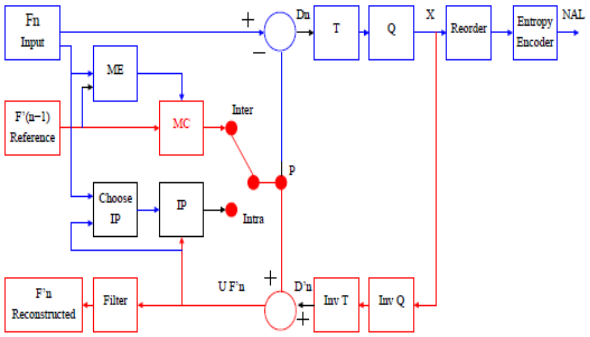

Figure 2: H.264 Video Encoder

The difference macro-block Dn is then transformed (T) and quantized (Q) to give X (a set of quantized transform coefficients). These quantized transform coefficients are re-ordered and then entropy encoded. The entropy encoded coefficients, together with side information required to decode the macro-block from the compressed bit-stream is passed to a Network Abstraction Layer (NAL) for transmission or storage.

3. Parallelization Opportunities

This section explores the various possibilities in the domain of parallel execution of video encoding applications.

Parallel Execution at Group of Pictures Level:
At Group of Pictures (GoP) level, several groups of consecutive frames are encoded simultaneously. It contributes to increased throughput. Authors in [13] defined a GoP as 15 consecutive frames. Computation resources were available in the form of cluster nodes. Consequently, a real time response is achieved but with a high latency.

Parallel Execution at Frame Level:
At frame level, frames are divided in several slices and processed in parallel. However, the drawback is limited scalability. Consequently, the real time response is achievable only under limited circumstances [14]. Similarly, authors in [12] and [16] implemented the H.264 video encoder using the concept of multithreading. [12] and [16] exploited the concepts of thread level parallelism and OpenMP programming model respectively.

Integration of GoP and Frame Level:
Integration of parallel execution at GoP and frame level ensures the compromise between throughput and latency [15], [21]. If the real time response is achieved, additional computational resources are used to parallelize GoP encoding in order to reduce latency.

Parallel Execution at Motion Estimation Level:
Parallel execution at motion estimation level is well-suited for hardware implementation. Authors in [17] and [18] classify the techniques at motion estimation level into three types: 1-D array, 2-D array and hierarchal architectures. However, these techniques deal with the fixed block size (16x16 or 8x8) and are not appropriate for the variable block size

Limitation of Existing Techniques:
Parallelization at GoP level, frame level, combination of GoP and frame level or motion estimation level is not suitable for heterogeneous reconfigurable platforms like the Molen architecture (case study in this article) due to following reasons:

- Space limitation of the exchange registers.
- Dynamic allocation of data buffer in reconfigurable processor of the Molen architecture is needed if the frame size is changed, which is not a good programming model for our target platform.
- In case of parallelization at motion estimation level, we have to pay huge overhead of data transfer between reconfigurable processor and main memory.
- These techniques are based on simulations performed in a PC environment (not in an embedded environment). Our objective in this article is to analyze the parallel performance without performing HW/SW co-simulation.

Macro-Block Level
Parallel execution at macro-block level is used for communication architectures. The constraints in these architectures are size of exchange registers and the overhead of data transfer between different processors and the main memory. Consequently, Molen architecture is communication architecture and we parallelize the H.264 video encoding algorithm at macro-block level. Examples of macro-block level parallelism can be found in [25] and [26].

Our interest in this article is to develop an analytical model for video encoding applications. One such example of parallel execution at macro-block level is presented in [7] that analyses the H.264 video encoding application for Cell processor. However, the following points make our work novel and useful:

- We have proposed a methodology for parallelization of video encoding applications, consisting of four steps. Then the proposed methodology is tested for H.264 video encoding application. It makes our work modular and thus the proposed methodology can be used for parallelization of other video encoding standards such as H.263, MPEG-4 etc. On the other hand, such a generic methodology is not found in [7] making the work dedicated to H.264 video encoders only.
- The proposed methodology is not specific to a particular platform. Molen architecture just serves as a case study. Consequently, the methodology can easily be applied to other heterogeneous platforms. On the other hand, the work in [7] exploits special features of Cell processor. For example, in order to execute a program using any of the 8 Synergistic Processor Elements (SPEs), it is essential to load the code and data on the 256KB local store (LS). If the sum of code and data memory requirements is larger than 256KB, code overlay technique should be used, which makes the parallelization difficult to achieve.

Parallel programming on heterogeneous reconfigurable architecture (such as Molen) is flexible as compared to Cell processor. It allows the design of parallel video encoders adapted to almost any requirement by selecting different number of custom computing units (described in Section 2 of this article).

4. Performance Methodology

This section proposes a performance analysis methodology for parallel execution of video encoding applications. The application performance is formulated with an equation. The main steps are shown in Figure 3.

Division of Source Code into Frame Data and Macro-block Data:

The source code in the form of C files is divided into two types: frame data processing and macro-block (MB) data processing. The example of frame data processing is the parsing of video bit-stream into MBs. The example of MB processing is the motion estimation step. The total execution time of the video encoding application is the sum of execution time for frame data processing and the execution time for MB processing. If a video frame consists of "N" MBs, MB data processing part is invoked "N" times for each of the frame data processing part.

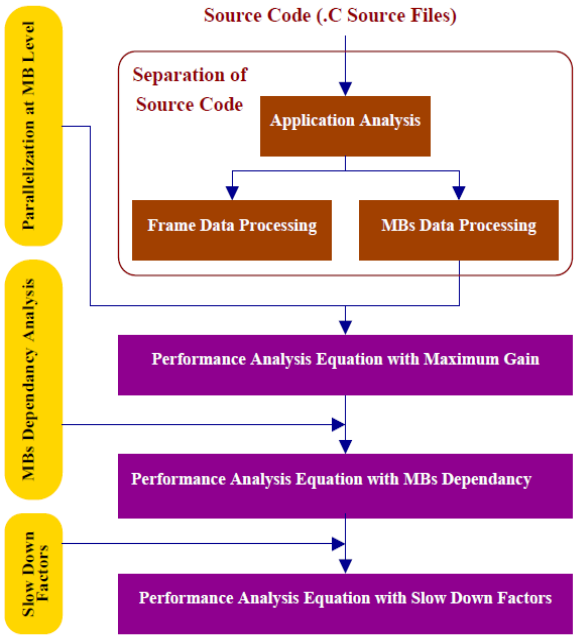

Figure 3: Proposed Analysis Methodology

Parallelization at Macro-block Level:
Once the source code is separated into frame data processing and MB data processing, the MB processing part is parallelized between different processing elements of the target heterogeneous reconfigurable architecture. The output is in the form of maximum possible performance gain with parallel execution. In this step, we do not consider the dependencies between different MBs. Section 6 of this article calculate the maximum possible performance gain for the H264 video encoding application on the Molen architecture.

Analysis of Dependencies in Macro-blocks:
The performance gain obtained in the second step is not realistic because of the data dependency between MBs. There is a time duration during which the

maximum parallelism is not achieved. In other words, there are some MBs which are not processed at maximum parallelism. Some MBs are processed before the maximum parallelism and some MBs are processed after the maximum parallelism. Dependency analysis is performed to formulate the sum of total time duration. Section 7 describes this step in more detail. The output of this step is the performance gain with parallel execution after considering the MBs dependency.

Performance Analysis with Slow Down Factors:
The performance gain obtained in the previous steps is further reduced by some slow down factors. These slow down factors cannot be determined at compile-time and are measured at run-time only. They depend on the characteristics of the input video bit-stream. The examples of slow down factors are the non-uniform execution time and the idle time for the reconfigurable processors.

5. Separation of Source Code

This section describes the first step of the proposed analysis methodology. We use X264 [22], [23] that is an open source implementation of H.264 video encoding algorithm. The profiling results with gprof profiling tool [24] are shown in Figure 4.

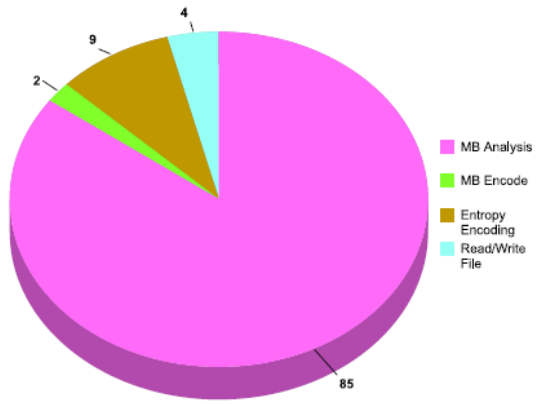

Figure 4: Results for x264 with GPROF Tool

The results in Figure 4 shows that computational intensive parts of the application are motion prediction and estimation (Here after we call it as MB Analysis). The other computational intensive part is encoding of macro-block (MB Encode).

The summary of source code separation is shown in Figure 5. The first block is "ReadFile" which parses the incoming video bit-stream into MBs. The processing in "ReadFile" is at frame level. The next two parts (blocks) are "MB Analysis" and "MB Encode". The processing in these blocks is at macro-block level. Again, at the end, the two blocks "Entropy Encoding" and "Write File" represent processing at frame level. For each frame, the macro-block processing section is invoked "N" times, if a frame consist of "N" macro-blocks. We can further pipeline the MB processing section itself between the MB Analysis and the MB Encode module as shown in Figure 5.

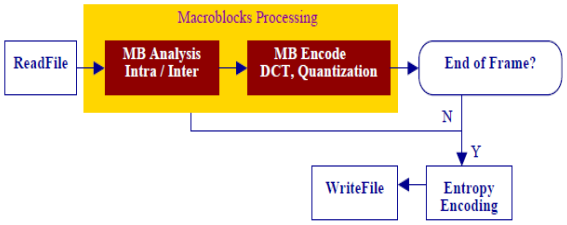

Figure 5: Separation of Source Code

6. Parallelization at Macro-block Level

Section 5 separated the source code into "frame data processing" and "MB data processing". This section describes the maximum performance gain with parallel execution without considering the MBs dependency.

Sequential Execution on PowerPC:
Figure 6 shows the sequential execution profile after pipelining the X264 encoding algorithm on the GPP of Molen architecture. PowerPC (PPC) is used as a GPP in our case study.

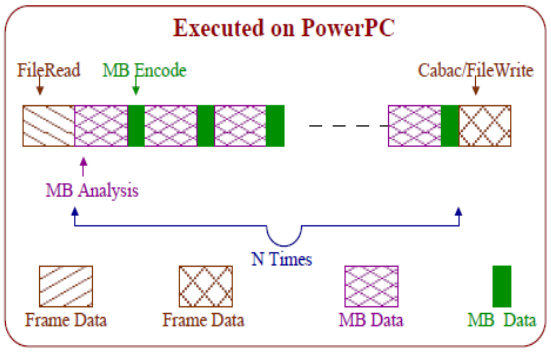

Figure 6: Sequential Execution on PowerPC

Figure 6 shows that the MB processing part (MB Analysis module and MB Encode module) is executed "N" times for each frame processing part (FileRead, FileWrite and Entropy Encoding modules). "N" represents the total number of MBs in a frame. The reference sequential execution in Figure 6 will be used for the performance comparison with parallel execution (described in Section 6.2).

Parallel Execution on PowerPC and Reconfigurable Co-processor:

In parallel execution, we move MB Analysis module to reconfigurable co-processor (RP) of the Molen architecture as shown in Figure 7. MB Encode module and frame data processing part are still executed on the PPC. However, MB Analysis module is shifted to the RP for parallel execution. Consequently, MB Analysis and MB Encoding overlap most of their executions to hide the MB Encoding time. On the other hand, the PPC should pay synchronization overhead between the two threads.

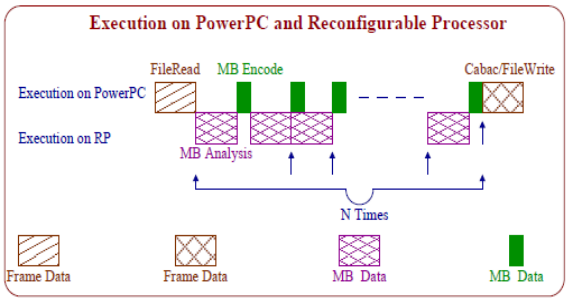

Figure 7: Parallel Execution Profile

Similar to work in [7], we need to define the following notations for analysis of parallel execution:

- "Tf": Sum of the execution time of the frame processing modules that are executed sequentially on the PowerPC.
- "Te": Execution time of the MB Encod, which is 2 % in Figure 4.
- "R": Performance ratio between the PowerPC and the RP. We assume that RP is faster than PowerPC by a ratio of 0.5.
- "P": Total number of co-processors used.
- "C": Overhead between the PowerPC and the RP for data transfer. We assume this value as 5 % of the total execution time.

Then, the equation for parallel execution time can be written as:

$$T=Tf+\left(\frac{1-Tf-Te}{P}\right)(R)+C \qquad (1)$$

As Tf = 0.04 (4 %) and Te = 0.02 (2 %). We have assumed that R = 0.5, P = 8 and C = 0.05 (5 %). Consequently, we get T = 0.14875. We know that the performance gain is the ratio of the execution time of the serial execution on a single processor to the parallel execution time. Therefore, the maximum performance gain with 8 co-processors (P=8) can be as large as 6.72 (1 divided by 0.14875), compared with the reference sequential execution on PPC. This gain is possible only if the macro-block processing is fully parallelizable. But it is not realistic so that we may not achieve that much performance gain due to the dependency of MBs (we will discuss it in Section 7) and slow down factors (we will discuss it in Section 8).

The work in [7] dis not compare the notations with the well-known Amdahl's law [19][20]. Before proceeding further, we compared our parallel execution notations with Amdahl's law.

Comparison of Parallel Execution with Amdahl's law:

According to Amdahl's law, speedup is the original execution time divided by an enhanced execution time. It implies that if we enhance a fraction "Par" of a computation by "P" number of processors, the overall speedup is:

$$\text{Speedup} = \frac{1}{(1-Par)+\frac{Par}{P}} \qquad (2)$$

Where "Par" in Equation 2 shows the code which can be run in parallel and "1-Par" represents the code which is executed sequentially. However, the Amdahl's law, described in Equation 2 is based on the assumption that the processors upon which the code executes are homogeneous. Therefore, it does not take into account the performance ratio between main processor and co-processors [19] [20]. Similarly, data transfer overhead between main processor and co-processors has not been taken into account. If "R" is the performance ratio and "C" is the overhead, the equation of speedup becomes:

$$\text{Speedup} = \frac{1}{(1-Par)+\left(\frac{Par}{P}\right)R+C} \qquad (3)$$

The denominator of Equation 3 is similar to the denominator of Equation 1. Therefore, the term"Par" in Equation 3 is represented by the term "1-Tf-Te" in Equation 1. We have also mentioned that "MB Encode" time is overlapped with "MB Analysis". Consequently, "1 - Par" in Equation 3 is represented by "Tf" in Equation 1.

This section provided a performance analysis equation without considering slowdown factors and compared the equation with Amdahl's law. The next two sections will add the slowdown factors of parallel execution. First, Section 7 will add slowdown factors due to dependency on previous MBs. Then, Section 8 will describe the input-dependent slow down factors.

7. Analysis of Macro-blocks

Dependency Analysis on Previous Macro-blocks:
Consider the example of a simple frame structure as shown in Figure 8.

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(1,8)	(1,9)	(1,10)	(1,11)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)	(2,8)	(2,9)	(2,10)	(2,11)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)	(3,8)	(3,9)	(3,10)	(3,11)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)	(4,8)	(4,9)	(4,10)	(4,11)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)	(5,8)	(5,9)	(5,10)	(5,11)
(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)	(6,8)	(6,9)	(6,10)	(6,11)

Figure 8: A simple frame that consists of 66 MBs

The frame consists of 66 macro-blocks, where MBs are indexed by (a,b). During the execution of MB analysis, there is a dependency between MBs [22]. The analysis of an MB needs to refer to the analysis results of some other MBs as shown in Figure 9. For example, in order to obtain the analysis results for "MB(a,b)", we need to refer to the analysis results of "MB(a-1,b)", "MB(a-1,b+1)" and "MB(a,b-1)".

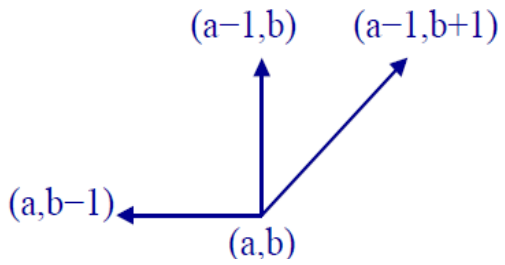

Figure 9: Dependency on previous MBs

Initially two MBs, "MB(1,1)" and "MB(1,2)" are analyzed sequentially. Suppose "t" be the execution time of one MB analysis. After the time "2t", two MBs, "MB(1,3)" and "MB(2,1)", can be concurrently analyzable. After the time "4t", three MBs are concurrently analyzable, and so on. Thus "N" macro-blocks are not fully parallelizable for their processing. Let us estimate the performance gain from parallel execution of macro-blocks processing. First we calculate the potential parallelism of the application with the following equation:

$$\text{PotentialParallelism} = \min\left(\frac{cols}{2}, rows\right) \qquad (4)$$

Equation 4 consists of two terms: The first term shows the total number of columns divided by 2. For example, the number of columns is 11, then this term will be equal to 5.5. However, the number of columns should be natural numbers. Consequently, we take this value equal to 6. Similarly, if the total number of columns is 9, we take this value as 5 and so on. The second term represents the number of rows. Finally, the potential parallelism will be equal to the minimum of the two terms. We calculate the maximum number of concurrently analyzable macro-blocks by Equation 5:

$$n = \min(\text{PotentialParallelism}, P) \qquad (5)$$

Where "n" is the maximum number of concurrently analyzable macro-blocks and "P" is the total number of co-processors (RPs) used. If potential parallelism is larger than the number of co-processors, the maximum number of concurrently analyzable MBs (represented by n) is limited to "RPs".

Formulation of Maximum Parallelism:
Equation 5 shows the maximum number of concurrently analyzable macro-blocks. Mathematically, "2(n-1)" time units are consumed before achieving the maximum parallelism.

Similarly, "2(n-1)" time units are consumed at the end of the profile. Consequently, the total number of time units consumed before and after the maximum parallelism are "4(n-1)". The total execution time consumed for the processing of macro-blockss before and after the maximum parallelism is represented by T1 and is given by the following equation:

$$T1 = \left[\frac{1-Tf-Te}{N}(R)\right] * [4(n-1)] \qquad (6)$$

The term "4(n-1)" in Equation 6 shows the sum of time durations before and after the maximum parallelism. The remaining MBs can be processed at full parallelism and its duration is computed with the term"N-2n(n-1)" divided by "n"(Where "N" is the total number of MBs in a frame). The execution time consumed for the processing of macro-blocks at maximum parallelism is represented by "T2" and is given by Equation 7.

$$T1 = [\frac{1-Tf-Te}{N}(R)] * [\frac{N-2n*(n-1)}{n}] \quad (7)$$

Formulation of Execution Time for Macro-block Analysis:
The total execution time of parallel execution for MB Analysis "T (analysis)" becomes,

$$T(analysis) = T1 + T2 \quad (8)$$

Putting the values in Equation 8, we get,

$$T(analysis) = [\frac{1-Tf-Te}{N}(R)] * [4(n-1 + \frac{N-2n*(n-1)}{n}] \quad (9)$$

After simplification, we get Equation 10.

$$T(analysis) = [\frac{1-Tf-Te}{N}(R)] * [\frac{N}{n} + 2 * (n-1)] \quad (10)$$

Equation 10 shows the total execution time of parallel execution of MB Analysis.

Formulation for Total Execution Time:
If we include the MBs dependency effects in Equation 1, the total execution time considering the MBs dependency becomes:

$$T(analysis) = [\frac{1-Tf-Te}{N}(R)] * [\frac{N}{n} + 2 * (n-1)] + C \quad (11)$$

Decrease in parallel execution due to the dependency of MBs is computed by dividing the second term of Equation 11 with the second term of Equation 1. The amount of slow down "S" due to the macro-blocks dependency is given by:

$$T(analysis) = \frac{P}{N} * [\frac{N}{n} + 2 * (n-1)] \quad (12)$$

After simplification, we get,

$$T(analysis) = \frac{P}{N} + \frac{P}{N} * [2 * (n-1)] \quad (13)$$

Equation 13 implies that the value of "S" depends upon two ratios: (1) the first ratio is the number of available co-processors (RPs) to the maximum parallelism. Greater this ratio is, larger will be the value of "S" and vice versa, (2) the second ratio is the number of available "RPs" to the total number of MBs in a frame. Figure 10 shows the value of "S" with the corresponding values of "P" and "n". The value of "N" for Figure 10 is taken as 300.

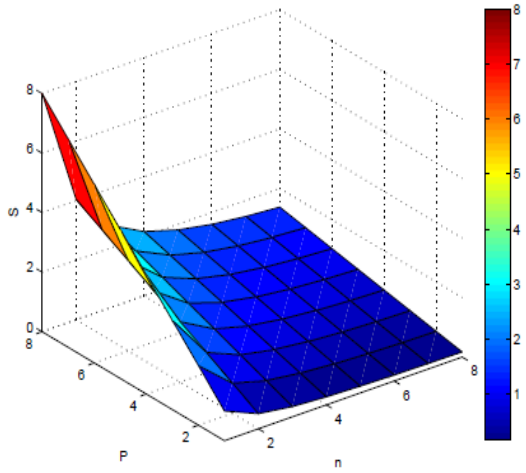

Figure 10: Slow down due to MBs dependency

This section provided the performance analysis equation after considering the dependency on previous MBs. It calculated the maximum number of concurrently analysable macro-blocks. Accordingly, it divided the total execution time of MB Analysis into: (a) the time duration during which maximum parallelism is achieved and (b) the time duration during which maximum parallelism is not achieved. Finally, it provided the slowdown due to dependency on previous MBs. The next section will provide the slowdown factors which cannot be determined at compile time and are dependent on the input video sequence.

8. Input Dependent Slow Down Factors

In this section, we describe two factors that hinder the speedup of parallel execution of H.264 video encoder. These slow down factor are dependent on the input video sequence. Therefore, it is not possible to compute them at compile-time.

Description of Non-uniform Execution Time:
The three types of frames are: "I", "P" and "B". The time variance of MB Analysis is not significant for "I" frames. However, the time variance for "P" and "B" frames is significant. Consequently, the MB Analysis of "P" and "B" frames consists of 5 steps as shown in Figure 11.

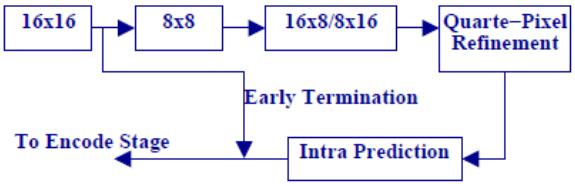

Figure 11: Steps of macro-block analysis

A"16x16" macro-block is first analyzed. If the difference between the motion estimated macro-block and the current macro-block is smaller than a threshold, the MB analysis is terminated, which is called early termination and saves significant portion of execution time. If the condition for early termination is not met, macro-block is divided into four "8x8" blocks for further analysis. If the computation cost of "8x8" analysis is smaller than that of "16x16" analysis, then next step "16x8" or "8x16" is performed. Once analysis of macro-blocks is done, quarter pixel refinement is performed. Finally, the last step is to perform Intra Prediction. It implies that the total execution time of MB analysis is not fixed and depends upon the input video sequence.

Description of Idle Time for Reconfigurable Processor (Co-processors):
Another slow down factor for parallel execution of MB analysis is the idle time for co-processors. For example, MB(2,1) is finished earlier due to early termination than MB(1,3) during [2T,3T]. As MB(2,2) and MB(1,4) depends on MB(1,3), the RP that executes MB(2,1) should wait until MB(1,3) is completed

Formulation for Total Execution Time with Slow Down Factors:
Let "D" represents all slow-down factors of parallel execution, the total execution time of H.264 video encoding application can be represented with the following equation:

$$T = Tf + \left(\frac{(1-Tf-Te)R}{N}\right)\left(\frac{N}{n} + 2(n-1)\right)(d) + C \qquad (14)$$

Where, the first term in Equation 14 represents the frame processing time. The second term shows the parallel execution of the MBs processing including dependency analysis and slows down factors of parallel execution. The third term shows the communication overhead between PowerPC and the reconfigurable processors.

9. Conclusion

This article proposed a performance analysis methodology of video encoding applications on heterogeneous reconfigurable architectures. The key benefit of the proposed methodology was the performance analysis of parallel execution prior to HW/SW co-simulation. H.264 video encoding application was used as a case study. The target platform was the Molen architecture. We formulated the performance in terms of maximum concurrently analyzable MBs, total number of MBs, performance ratio and data transfer overhead.

References

[1] Jason Howard et al. "A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling", IEEE Journal of Solid-State Circuits, Vol. 46, No. 1, pp. 173-183, 2011.
[2] Dawid Zydek and Henry Selvaraj, (2011) "Fast and efficient processor allocation algorithm for torus-based chip multiprocessors", Computer and Electrical Engineering, Vol. 37, No 1, pp. 91-105, 2011.
[3] Zhenyu Liu et al., "HDTV1080p H.264/AVC encoder chip design and performance analysis", IEEE Journal of Solid-State Circuits, Vol. 44, No. 2, pp. 594-608, 2009.
[4] J. Teich, "Hardware/software co-design: The past, the present, and predicting the future", Proceedings of the IEEE, Vol. 100, pp. 1411-1430, 2012.
[5] K. Pingali et al., "The tao of parallelism in algorithms", ACM Sigplan Notices, Vol. 47, No. 6, pp. 12-25, 2011.
[6] S. Zalan et al. "High-level multicore programming with C++", Computer Science and Inf. Systems, Vol. 9, No. 3, pp. 1187-1202, 2012.
[7] J. Park, S. Ha, "Performance analysis of parallel execution of H.264 encoder on the cell processor", In Proc. of IEEE/ACM Workshop on Embedded Systems, pp. 27–32, 2007.
[8] M. A. Shami and A. Hemani, "An improved self-reconfigurable interconnection scheme for a Coarse Grain Reconfigurable Architecture", IEEE Proceedings on 28th NORCHIP conference,

pp.1-6, Tampere, November 2010.
[9] R. Patel and A. Rajawat, "A survey of embedded software profiling methodologies", International Journal of Embedded Systems and Applications, Vol. 1, No. 2, 2011.
[10] "ITU-T Rec. H.264-ISO/IEC 14496-10 AVC, Document JVT-D157", Austria, July 2002.
[11] S. Vassiliadis et al., "The molen polymorphic processor.", IEEE Transactions on Computers, Vol. 53, No. 11, pp. 1363–1375, 2004.
[12] M. Alipour and H. Taghdisi, "Effect of thread level parallelism on the performance of optimum architecture for embedded applications", International Journal of Embedded Systems and Applications , Vol. 2, No. 1, 2012.
[13] A. Rodriguez, A. Gonzlez and M. P. Malumbres, "Performance evaluation of parallel MPEG-4 video coding algorithms on clusters of workstations", In Proceedings of the International Conference on Parallel Computing in Electrical Engineering, pp. 354–357, 2004.
[14] Cor Meenderinck et al. "Parallel scalability of H. 264", Proceedings of the first Workshop on Programmability, Issues for Multi-Core Computers. 2008.
[15] A. Rodriguez, A. Gonzalez & M. P. Malumbres, "Hierarchical parallelization of an H.264/AVC video encoder", In Proceedings of the International Symposium on Parallel Computing in Electrical Engineering, pp. 363-368, 2006.
[16] Y. Chen, X. Tian, S. Ge and M. Girkar, "Towards efficient multi-level threading of H.264 encoder on intel hyper-threading architectures", In Proceedings of the 18th International Parallel and Distributed Processing Symposium, pp. 63–70, 2004.
[17] C. Y. Cho, S. Y. Huang and J. S. Wang, "An embedded merging scheme for VLSI implementation of H.264/AVC motion estimation" In Proc. of the IEEE Conf. on Image Processing, Vol. 3, pp. III - 1016-19, 2005.
[18] C. Y. Cho, S. Y. Huang and J. S. Wang, "An embedded merging scheme for H.264/AVC motion estimation", In Proc. of the IEEE International Conference on Image Processing, Vol. 1, pp. I - 909-12, 2003.
[19] J. M. Paul & B. H. Meyer, "Amdahl's law revisited for single chip systems" In International Journal of Parallel Programming, Vol. 35, No. 2, pp. 101-123, 2007.
[20] M. D. Hill, "Amdahl's Law in the Multi core Era" In Proc. of the International Symposium High Performance Computer Architecture, Keynote, pp. 33-38, 2008.
[21] W. Haitao, J. Yu, J. Li, "The design and evaluation of hierarchical multi-level parallelisms for H.264 encoder on multi-core architecture", Computer Science and Information Systems, Vol. 7, No. 1, pp. 189-200, 2010.
[22] X264 video encoder, Available: http://www.videolan.org/developers/x264.html.
[23] JM Reference, Available: http://iphome.hhi.de/suehring/tml/index.htm.
[24] GNU GCC Tool Suit, [Online]. Available: http://gcc.gnu.org
[25] S. Sun, D. Wang and S. Chen, "A Highly Efficient Parallel Algorithm for H.264 Encoder Based on Macro-Block Region Partition", Lecture Notes in Computer Science, Vol. 4782, pp. 577-585, 2007.
[26] Z. Wang et al.,"Novel Macro-Block Group Scheme of AVS Coding for Many-Core Processor", Journal of Signal Proce. Systems, Vol. 65, No. 1, pp. 129-145, 2011.

Muhammad Rashid received the Bachelor's degree in electrical engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2000, the Master's degree in embedded systems design from the University of Nice, Sophia-Antipolis, France, in 2006, and the Ph.D. degree in embedded systems design from the University of Bretagne Occidentale, Brest, France, in 2009. Currently, he is an Assistant Professor with the Computer Engineering Department, Umm Al-Qura University, Mecca, Saudi Arabia.