FORMULATION OF VIDEO ENCODING APPLICATIONS FOR HETEROGENEOUS RECONFIGURABLE ARCHITECTURES

Muhammad Rashid
Department of Computer Engineering
College of Computers & Information Systems
Umm Al-Qura University
Makkah, Saudi Arabia
mfelahi@uqu.edu.sa

Abstract— **Heterogeneous reconfigurable architectures are being used for video encoding applications. During early stages of the design flow, performance estimation due to parallel execution of video encoding applications on heterogeneous platforms is a critical requirement. In this article, we formulate the performance of parallel execution of H.264 video encoding application on a heterogeneous reconfigurable architecture. The formulation process starts with the analysis of video encoding application and separates the application into frame data processing and macro-block processing. Parallelization is done at macro-block level and performance gain is derived with an equation.**

Keywords-analysis; performance; parallel; estimation, HW/SW co-simulatio, partitioning, mapping

I. INTRODUCTION

The emerging multimedia standards are becoming more and more complex [1]. An example of this phenomenon is the continuous growing effort of developing video encoding standards (such as H.264) with higher compression efficiency, better quality of service and more functionality. Consequently, the complexity of video encoding standards is increasing day by day. As the standard becomes more complex, the encoding process requires much more computation power than previous standards. A number of mechanisms are required to improve the speed of video encoder.

Heterogeneous reconfigurable architectures may provide enough computational power for video encoding applications. An example of heterogeneous architecture is the combination of general purpose processor with reconfigurable hardware such as FPGA (Field Programmable Gate Arrays) [2]. Partitioning video encoding application among software running on a general purpose processor and reconfigurable hardware (FPGA) improves the performance. However, there is no generally accepted partitioning methodology to separate applications into hardware and software execution [3].

Performance improvement by parallel execution on heterogeneous reconfigurable platforms depends upon effective mapping of applications on architectures [4]. In addition to effective mapping, potential parallelism of the application itself is also critical [15].

A HW/SW co-simulation step is performed to estimate the performance of parallel execution [16]. However, it requires either the actual hardware platform or virtual prototype of the 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 project is a real bottleneck. Alternative solution is the performance analysis of parallel execution by formulating the application performance on the target architecture with an equation.

This article formulates the performance of video encoding applications for heterogeneous reconfigurable architectures. By formulation, we mean the derivation of performance gain by parallel execution with an equation. The application is analyzed and separated into frame data processing and macro-block processing parts. Once the application is analyzed, an application parallelization approach at macro-block level is presented. First, the performance of parallel execution is derived without considering the macro-blocks dependency and then hindrance due to macro-blocks dependency is added to get the final equation.

H.264 video encoding application [5] serves as a case study and the target platform is the Delft workbench [6] (a heterogeneous reconfigurable platform). H.264 is getting wider acceptance due to its high-quality coding of video contents at very low bitrates. Delft Workbench supports integrated HW/SW co-design starting from profiling and partitioning to synthesis and compilation. It is based on Molen programming paradigm.

The rest of this article is organized as follows: Section II reviews the background information on Molen architecture and H.264 video encoding application. Section III provides the related work in parallelization of video encoding applications. Section IV separates the H.264 video encoding application into frame data and macro-block processing. Section V describes the parallelization at macro-blocks level. Section VI describes the dependency analysis of macro-blocks for parallel execution. Finally, we conclude the article in Section VII.

II. BACKGROUND INFORMATION

In this section, we briefly describe the target heterogeneous architcture and video encoding application.

A. *Target Architecture: Delft Workbench*

The Delft Workbench is based on Molen Programming Paradigm [6] to speed-up application execution by implementing its most critical functions as hardware accelerators, referred to as Custom Computing Units (CCUs). The Molen machine organization that supports the Molen Programming Paradigm is described in Figure 1.

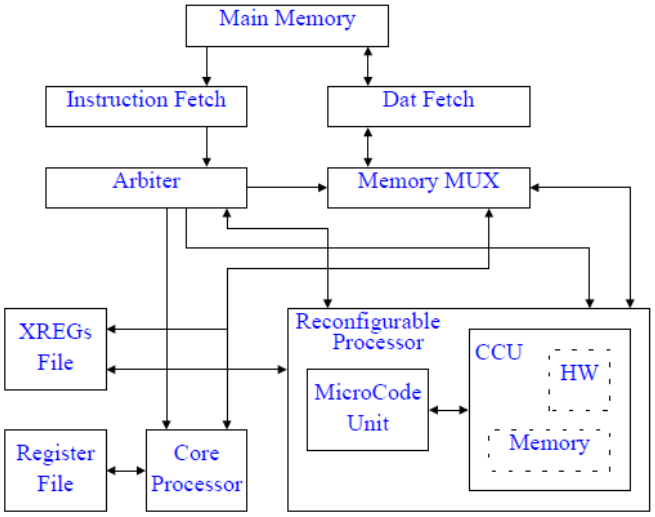

Figure 1. The Molen machine organization

The main parts are the general purpose processor (PowerPC in this case study), the reconfigurable co-processor (RP) and the Arbiter. The GPPs Instruction Set Architecture (ISA) is extended, in order to control the hardware accelerators (Reconfigurable Instructions - RI). The Arbiter fetches the applications instructions from the main memory. It partially decodes each one of them and checks whether it belongs to the standard or to the extended ISA and arbitrates them to the corresponding processor. The data transfer between the GPP and the RP is performed through exchange registers (XREG), organized in a register file. Figure 2 gives an example of data exchange through the XREGs.

The pragma annotation in Figure 2 recognizes the function for hardware implementation. During the program compilation, the memory addresses of *X* and *Y* are stored in XREGs. When the Arbiter detects an RI, it will send the appropriate reconfigurable microcode address to the MicroCode unit. The MicroCode unit is responsible for reading the microcode address and subsequently fetching the respective CCUs microcode. It enables the execution of addressed CCU. In the same time, the arbiter stalls the GPP. The CCU then reads the XREGs. When the CCU completes its computations, the returned value *res* is stored back to an XREG defined by the user prior to the compilation. The Arbiter detects that the CCU has finished and resumes the GPP. As the program execution is continued, the returned value is read and used.

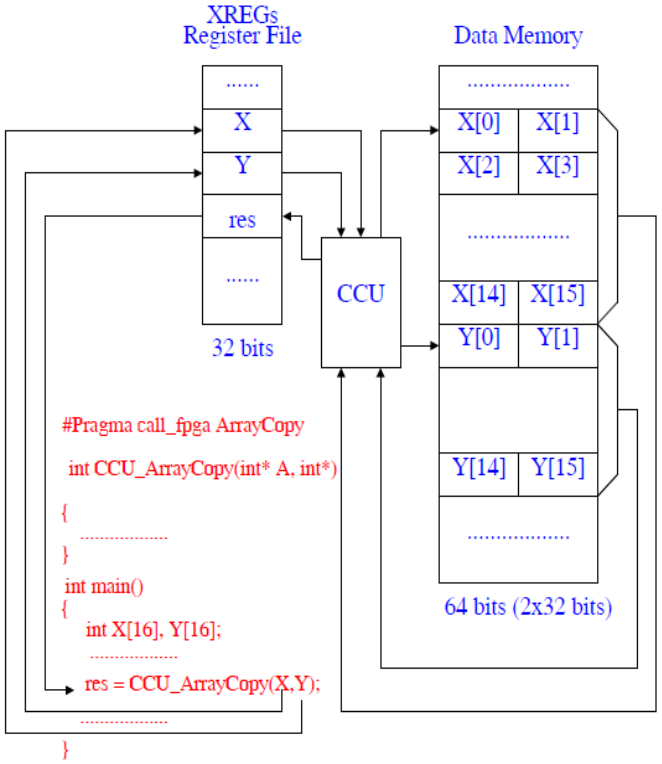

Figure 2. Data exchange between GPP and CCU through XREGs

B. *H.264 Video Encoding Application*

Block diagram of H.264 standard [5] is shown in Figure 3.

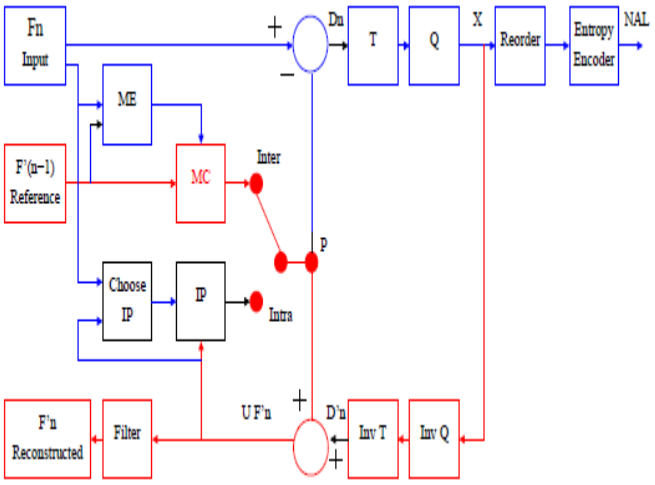

Figure 3. Block diagram of the H.264 video encoder

The input frame Fn is processed in units of macro-block (corresponding to 16x16 pixels in the original image). Each macro-block is encoded in intra or inter mode. In either case, a prediction macro-block P is formed. In Intra prediction mode, P is formed from samples in the current frame that have previously encoded, decoded and reconstructed. In Inter mode, P is formed by motion-estimated (ME) and motion-compensated (MC) prediction from one or more reference

frame(s). The prediction P is subtracted from the current macro-block to produce a residual or difference macro-block Dn. The difference macro-block is transformed (T) and quantized (Q) to give X (a set of quantized transform coefficients). These coefficients are re-ordered and entropy encoded. The entropy encoded coefficients, together with side information required to decode the macro-block (such as the macro-block prediction mode, quantizer step size, motion vector information etc.) from the compressed bit-stream is passed to a Network Abstraction Layer (NAL) for transmission or storage.

III. RELATED WORK

This section presents related work in the domain of parallel execution of video encoding applications. There are several approaches to parallelize the video encoding application. The most important choices to perform parallel execution are based on Group of Pictures (GoP) Level [7], Frame or Slice Level [8], Combination of GoP and Frame Level [9], [10] and Motion Estimation Level [11], [12].

A. Parallelization at Group of Pictures (GoP) Level

Each video sequence is composed of a series of Groups of Pictures (GoP). A GoP specifies the order in which frames are arranged. Parallelization at GoP level encodes simultaneously several groups of consecutive frames. Processing several GOPs in parallel contributes to increased throughput. Authors in [7] defined a GoP as 15 consecutive frames following the IBBPBBP..... coding pattern. Considering the availability of computation resources in the form of cluster nodes, a real time response is achieved but with a high latency.

B. Parallelization at Frame Level

A frame is composed of a series of slices whereas a slice is composed of a series of macro-blocks. In order to reduce the latency found in GoP level parallelism, frames are divided in several slices and processed in parallel. However, it gives limited scalability and the real time response is achievable only under limited circumstances [8]. A multi-threaded implementation of the H.264 encoder is presented in [10]. It exploits multiple levels of parallelism including frame level and slice level based on OpenMP programming model. The parallelism is exploited by adding few pragmas in the serial code. A parallelizing compiler converts the serial code to multi-threaded code automatically. Trade-offs of using different parallelism in video codec and the final implementation scheme have been illustrated.

C. Combination of GoP and Frame Level

GoP parallelism gives good speedup but imposes very high latency. On the other hand, frame parallelism gets less efficiency but low latency. A compromise between speedup and latency is obtained in [9] by combining GoP and frame level approaches. When the real time response is achieved, that is throughput is equal to the frame rate, additional computational resources are used to parallelize GoP encoding in order to reduce latency. Cluster configuration is adjusted by: (1) setting up the number of processor groups or parallel encoded GoPs and (2) the number of processor in a group or number of slices in a frame. Depending on the application requirements, an adequate balance between throughput and latency is achieved.

D. Disadvantages of GoP and Frame Level Parallelism

Parallelization at GoP level or frame level does not fit for an heterogeneous platform like the Molen architecture (case study in this article) due to space limitation of the "exchange registers". In addition to the size of 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.

E. Motion Estimation Level

Parallel execution at motion estimation level is the most popular technique for hardware implementation. Authors in [11] review some existing VLSI motion estimation architectures and classify them into three types: 1-D array, 2-D array and hierarchal architectures. However, they are all designed to deal with the fixed block size (16x16 or 8x8) and are not appropriate for the variable block size motion estimation in the emerging H.264/AVC standard. Furthermore, If we use the same parallel execution on the Molen architecture, we have to pay huge overhead of data transfer between reconfigurable processor and main memory.

F. Limitations of Existing Techniques

The limitations of existing techniques are size of exchange registers, dynamic allocation of data buffer, and overhead of data transfer between reconfigurable processor and the main memory. Consequently, we parallelize the video encoding applications at macro-block (MB) level. However, parallelization at MB level requires the separation of input application into frame data processing and MB data processing.

Next section of this article analyzes the open source implementation of H.264 video encoder, known as X264, and separates it into frame data and MB data.

IV. ANALYSIS OF H.264 VIDEO ENCODER

X264 [12] is an open source implementation (in C language) of H.264 video encoding application. In the following, we analyze the X264 code.

A. Hierarchical Structure of X264 Encoder

Figure 4 shows hierarchical structure of X264 during its execution. The execution starts from the "main" function in "x264.c". It calls other functions in "muxers.c", "common.c" and "encoder.c". Functions in "muxers.c" provide container format and the functions in "common.c" are used for parsing the command line and allocating dynamic memory functions. Functions in "encoder.c" are basic encoding functions. Figure 4 shows that "encoder.c" contains top level functions to perform encoding.

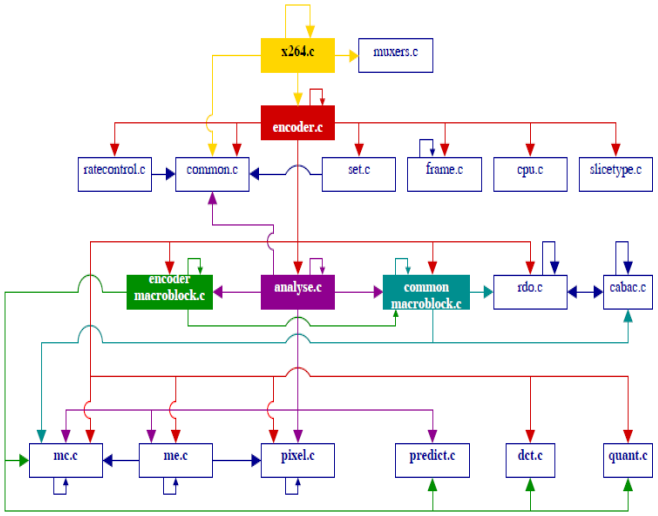

Figure 4. Hierarchical structure of X264 encoder in terms of source files

There is a sequential execution from the function "main" to the function "x264_slice_write" as shown in Figure 5.

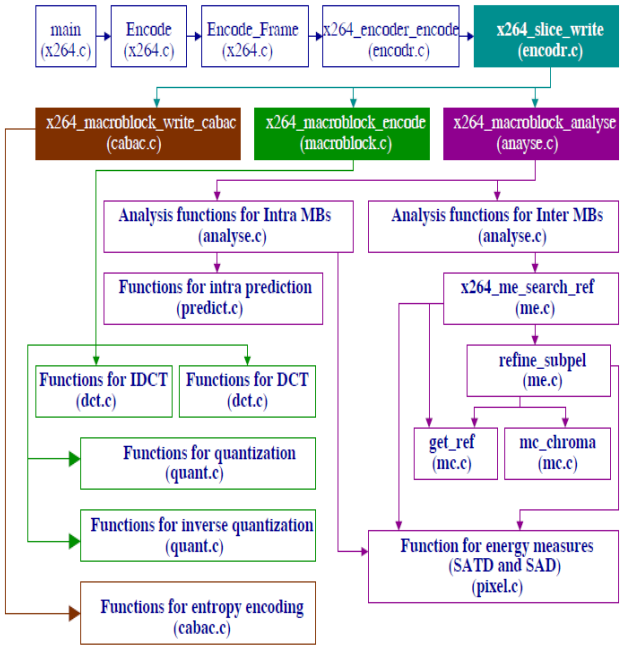

Figure 5. Functions hierarchy in the X264 video encoding source code

Up to this point, the frame is transformed into MBs. From here, the application is divided into three main parts:

- The function "x264_macroblock_analyse" and its sub-functions perform analysis of inter or intra macro-blocks.
- The function"x264_macroblock_encode" and its sub-functions perform transformation and quantization.
- The function "x264_macroblock_write_cabac" and its sub-functions perform entropy encoding.

"x264_macroblock_analyse" is further sub-divided according to MB type (inter or intra). Two processes are important for Inter MB analysis: First is the motion estimation process. It is the process of determining motion vectors to a suitable reference area in a previously coded frame with the help of two energy measures: Sum of Absolute Transform Difference (SATD) and Sum of Absolute Difference (SAD). The second important process in inter MB analysis is the motion compensation. It uses motion vectors to transform a reference picture into current picture. The functions "get_ref" and "mc_chroma" are responsible for motion compensation.

As far as the analysis of intra MBs is concerned, it uses SAD and SATD functions for motion estimation. In addition to this, it calculates intra prediction modes to compute the intra prediction. Intra prediction means that the samples of a MB are predicted by using information of already transmitted MBs of the same frame. There are two types of intra prediction. The first type is for 4x4 MBs while the second type is for 16x16 MBs. "x264_macroblock_encode" is further subdivided into four types as shown in Figure 5. These types are: Functions for Discrete Cosine Transform (DCT), Functions for Inverse DCT (IDCT), Functions for quantization and Functions for inverse quantization. "x264_macroblock_write_cabac" have merged all the functions related to entropy encoding in one function.

To summarize, this section has developed the following points:

- The three major categories in X264 source code are:
  - Macro-block Analysis (MB Analysis)
  - Macro-block Encode (MB Encode)
  - Entropy Encoding
- MB Analysis includes motion estimation, motion compensation and intra prediction.
- MB Encode contains DCT, quantization, inverse DCT and inverse quantization.

B. Profiling Results

The profiling results of X264 video encoding application with gprof [14] are shown in Figure 6.

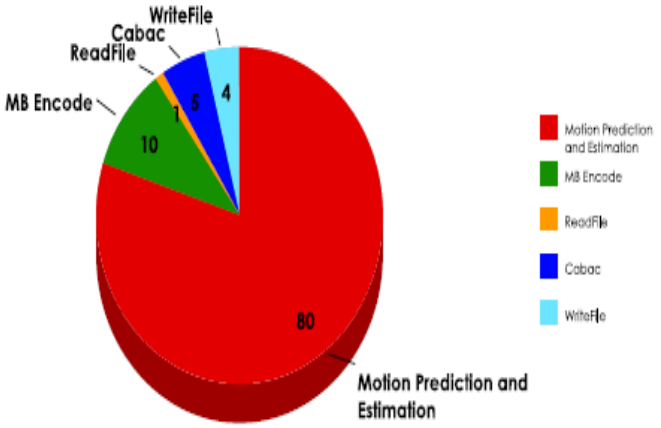

Figure 6. X264 video encoding application results with GPROF

C. Summary of Analysis Results

The summary of our analysis results is shown Figure 7. The first block (ReadFile) parses the incoming video bit-stream into MBs. The processing in ReadFile block is at frame level. The next two blocks are MB Analysis and MB Encode. The processing in these blocks is at MB level. Finally, the two blocks CABAC (Context-adaptive binary arithmetic coding) and WriteFile contain processing at frame level.

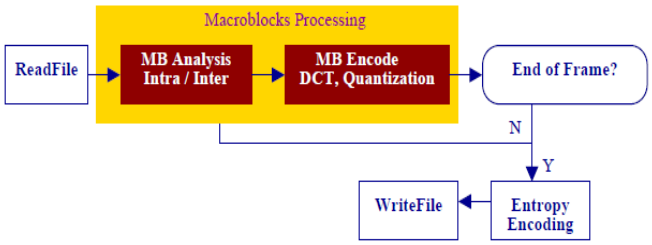

Figure 7: Separation of source code into frame data and MB processing

V. Parallelization at Macro-block Level

Section IV separated the source code into "frame data processing" and "MB data processing". In this section, we take MB data processing part and describe the maximum performance gain with parallel execution without considering the MBs dependency.

During sequential execution, the entire MB data processing (MB Analysis as well as MB Encode) is executed on the GPP of Molen architecture. During parallel execution, we move MB Analysis module to reconfigurable co-processor (RP) of the Molen architecture.

Consequently, we define the following notations for analysis of parallel execution:

- "Tf": Execution time for frame data processing. It sums up to 4 % in Figure 6.
- "Te": Execution time of MB Encod, which is 2 % in Figure 6.
- "R": Performance ratio between the GPP and the RP. We assume that RP is faster than GPP by a ratio of 0.5.
- "P": Total number of RP processors used.
- "C": Data transfer overhead between the GPP and the RP. Let us take this value as 5 % of the total execution time.

Then, a simple guess of the expected performance becomes:

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

We know that Tf = 0.04 (4 %) and Te = 0.02 (2 %). We have assumed that R = 0.5, P = 8 and C = 0.05 (5 %). Putting all these values in Equation 1, we get T = 0.14875. Since 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 RPs (P=8) can be as large as 6.72 (1 divided by 0.14875), compared with the reference sequential execution on GPP.

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

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

Where "Par" in Equation 2 represents the code which can be run in parallel and "1-Par" represents the code which is executed sequentially. If "R" is the performance ratio and "C" is the data transfer overhead, the equation of speedup becomes:

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

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

VI. Dependency Analysis of Macro-blocks

Section V derived a performance analysis equation without considering the MBs dependency. This section adds MBs dependency effects in Equation 1.

A. Dependency on Previous Macro-blocks

We explain the MBs dependency phenomenon by taking 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 macro-blocks

The frame consists of 11x6 MBs where MBs are indexed by (a,b). During MB analysis execution, there is dependency between MBs. The analysis of an MB needs to refer to the analysis results of some other MBs. To analyze "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)". In other words, for the analysis of an MB, we have to look in three different directions as shown in Figure 9. We must have the analysis results of MBs at these three directions.

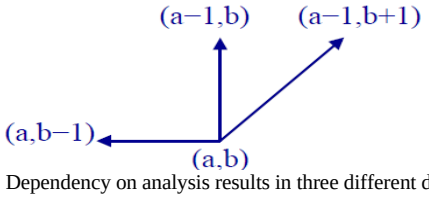

Figure 9: Dependency on analysis results in three different directions

Therefore, the profile of MB processing becomes as shown in Figure 10. Initially two MBs, "MB(1,1)" and "MB(1,2)" should be analyzed sequentially. Let "t" be the execution time of one MB analysis. After "2t", two MBs, "MB(1,3)" and "MB(2,1)", can be concurrently analyzable. After "4t", three MBs are concurrently analyzable, and so on. Thus "N" macro-blocks are not fully parallelizable for their processing.

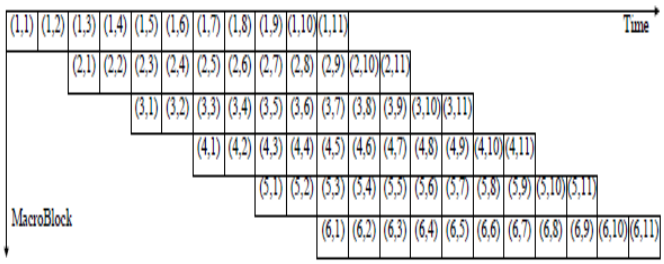

Figure 10: Macro-blocks processing profile after considering the dependency

Let us estimate the performance gain from parallel execution of MBs processing. First we compute the potential parallelism of the application.

$$PotentialParallelism = min(\frac{cols}{2}, rows) \quad (4)$$

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

$$n = min(PotentialParallelism, P) \quad (5)$$

Where "n" is the maximum number of concurrently analyzable MBs and "P" is the number of RPs used. If potential parallelism is larger than the number of RPs, the maximum number of concurrently analyzable MBs (n) is limited to "RPs". Figure 10 shows that this number is 6 for the frame of Figure 8. After achieving the maximum parallelism, the parallelism is again decreased at the tail of the profile, to make the profile bi-symmetric.

B. *Computing Maximum Parallelism during Macro-block Analysis*

Equation 5 represents the maximum number of concurrently analyzable MBs. Now the two questions arise: (1) what are the number of time units consumed before and after maximum parallelism and (2) What are the number of MBs processed during the time units when maximum parallelism is not achieved. Mathematically, "2(n-1)" time units are consumed before achieving the maximum parallelism. Similarly, "2(n-1)" time units are consumed at the tail of the profile. Therefore, the total number of time units consumed before and after the maximum parallelism are "4(n-1)". The total number of MBs processed before the maximum parallelism are "n(n-1)". Similarly, "n(n-1)" MBs are processed after the maximum parallelism. So the total number of MBs processed before and after the maximum parallelism are "2n(n-1)". The execution time consumed for the processing of MBs before and after the maximum parallelism is represented by T1 and is given by the following equation:

$$T1 = [\frac{1 - Tf - Te}{N}(R)] \times [4(n-1)] \quad (6)$$

The term "4(n-1)" in Equation 6 indicates the sum of time durations before and after the maximum parallelism. The maximum parallelism is achieved after we spend "2(n-1)" time units in the beginning of the profile. The same amount of time should be spent at the tail of the profile. During that duration, as many as "2n(n-1)" MBs are analyzed. For example, in Figure 10, the value of "n" is 6. Therefore, 20 time durations are consumed before and after the maximum parallelism. The total number of MBs processed during this time duration is 60. The remaining MBs can be processed at full parallelism and its duration is formulated 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 MBs at maximum parallelism is represented by "T2" and is given by Equation 7.

$$T2 = [\frac{1 - Tf - Te}{N}(R)] \times [\frac{N - 2n \times (n-1)}{n}] \quad (7)$$

For example, in Figure 10, the value of "n" is 6. Therefore, 1 time durations is consumed at maximum parallelism.

C. *Execution Time for Macro-block Analysis*

The total execution time of parallel execution of MB Analysis "T(analysis)" becomes,

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

Putting the values in Equation 8, we get,

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

On further simplifying Equation 9, we get Equation 10.

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

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

D. Total Execution Time of Video Encoding Application

The total execution time considering the MBs dependency becomes:

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

Decrease in parallel execution due to the dependency of MBs is obtained by dividing the second term of Equation 11 with the second term of Equation 1. Slow down "S" due to the MBs dependency is given by:

$$S = \frac{P}{N} \times [\frac{N}{n} + 2 \times (n-1)] \qquad (12)$$

E. Summary of Performance Analysis

This article has introduced the notations of potential parallelism, concurrently analyzable MBs and slows down factors for parallel execution of H.264 video encoding application. In the following, we summarize the key points:

- The maximum number of concurrently analyzable MBs (represented by n) is limited either by the amount of potential parallelism or the number of available configurable processors (represented by P).
- The total number of time units consumed without maximum parallelism in MB Analysis are computed as 4(n-1), while the total number of MBs processed during this time duration are computed as 2n(n-1).
- The slow down due to MBs dependency depends upon two ratios P/n and P/N. Where N is the total number of MBs in a frame.

VII. CONCLUSIONS

This article formulated the H.264 video encoding application on a heterogeneous reconfigurable architecture. Application source code was separated into frame data and macro-block processing. Once the application was analyzed into frame data and macro-block data, parallelization was performed at macro-block level. The expected performance with parallel execution was represented by an equation. We formulated the performance in terms of maximum concurrently analyzable MBs, total number of MBs, performance ratio and data transfer overhead between different processors.

This article described macro-block dependency that hinders the speedup of parallel execution of H.264 video encoder. In addition to hindrances created by macro-block dependencies, there are some other slow down factor that are dependent on the input video sequence and it is not possible to compute them at compile-time. Future work may focus on the inclusion of input-dependent slow down factors in performance equation.

REFERENCES

[1] Yun Q. Shi and Huifang Sun, "Image and video compression for multimedia engineering: fundamentals, algorithms, and standards," Second Edition, Pages 576, Massachusetts, USA, March 2008.

[2] Benjamin Krill et al., "An efficient FPGA-based dynamic partial reconfiguration design flow and environment for image and signal processing IP cores,", Signal Processing: Image Communication vol. 25, no. 5 , 2010, pp. 377-387.

[3] Peter Marwedel at al., "Mapping of applications to MPSoCs", In Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp. 109–118, Taipei, Taiwan, 2011.

[4] Michael I. Gordon, William Thies, and Saman Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelismin stream programs", In SIGARCH Computer Architecture News, vol. 34, no 5, 2006, pp. 151–162.

[5] H.264 Standard, "ITU-T recommendation H.264 ISO/IEC international standard, ISO/IEC 14496-10, video coding, 2005.

[6] Stamatis Vassiliadis, Stephan Wong, Georgi Gaydadjiev, Koen Bertels, Georgi Kuzmanov and Elena Moscu Panainte, "The molen polymorphic processor.", IEEE Transactions on Computers, vol. 53, no. 11, 2004, pp. 1363–1375.

[7] 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, 2004, pp. 354–357.

[8] A. Rodriguez, A. Gonzlez & 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, 2004, pp. 354–357.

[9] 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, 2006, pp. 363-368.

[10] Yen-Kuang Chen, Xinmin Tian, Steven Ge and Milind 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, 2004, pp. 63–70.

[11] Chuan-Yu Cho, Shiang-Yang Huang, Jenq-Neng Hwang, & Jia-Shung Wang, "An embedded merging scheme for VLSI implementation of H.264/AVC motion estimation" In Proceedings of the IEEE Int. Conference on Image Processing, Vol. 3, 2005, pp. III - 1016-19.

[12] Chuan-Yu Cho, Shiang-Yang Huang and Jia-Shung Wang, "An embedded merging scheme for H.264/AVC motion estimation", In Proceedings of the IEEE International Conference on Image Processing, Vol. 1, 2005, pp. I - 909-12.

[13] X264, http://www.videolan.org/developers/x264.html.

[14] J. Fenlason and R. Stallman. "The GNU profiler."

[15] Muhammad Rashid, Damien Picard and Bernard Pottier, "Application Analysis for Parallel Processing", In Proceedings of the 11th EuroMicro Conference on Digital System Design, Architectures, Methods and Tools (DSD'08), Parma, Italy, September 2008, pp. 633–640.

[16] Jurgen Teich, "Hardware/software codesign: The past, the present, and predicting the future", Proceedings of the IEEE, Vol. 100, Special Centennial Issue, 2012, pp. 1411-1430.