A High Level Design based on Performance Estimation Methodology for Reconfigurable Architectures

Muhammad Rashid
Thomson Silicon Components, Rennes, France
e-mail: muhamad.rashid@thomson.net

Abstract

This paper presents a high level design of multimedia applications based on performance estimation methodology for heterogeneous reconfigurable architectures. We formulate the performance estimation of a multimedia application on a target architecture to propose a high level design. As a case study, we propose a high level design for the H.264 encoding application on the Delft workbench, which is an heterogeneous reconfigurable architecture.

1. INTRODUCTION

Partitioning an embedded application among software running on a microprocessor and reconfigurable hardware improves the performance of embedded systems. However, there is no generally accepted methodology to separate applications onto hardware and software execution [1]. In the heterogeneous system context, this partitioning problem becomes more complicated due to the exponential growth of VLSI (Very Large Scale Integration) technology [2].

At the same time multimedia standards become more and more complex. It implies that the encoding process requires much more computation powers than previous standards. High level estimation of embedded systems allows rapid performance evaluation of different design parameters, application to processor mapping and hardware-software partitioning.

This paper present a high level design based on performance estimation methodology for video encoding applications on heterogeneous reconfigurable architectures. We analyze the application and presents an application parallelization approach for the target platform. H.264 video encoding application [3] serves as a case study and the target platform in this paper is the Delft workbench [4] (an heterogeneous reconfigurable platform). H.264 is getting wider acceptance due to its high-quality coding of video contents at very low bitrates than the previous standards such as, H.263 and MPEG-4 [5]. The Delft Workbench supports integrated hardware-software co-design starting from profiling and partitioning to synthesis and compilation. It is a semiautomatic tool platform targeting the Molen polymorphic organization and supporting the Molen programming paradigm. Considering the communication architecture of the target architecture, we parallelize the H.264 encoding algorithm at macroblock level.

The rest of this paper is organized as follows: Section 2 reviews the background information on the Delft workbench and the H.264 encoding application. Section 3 presents the proposed performance estimation methodology. Section 4 describes the H.264 application analysis. Section 5 presents a high level design for H.264 based on analysis and performance estimation results. Section 6 concludes the paper.

2. BACKGROUND: ARCHITECTURE AND APPLICATION

Before discussing our proposed performance estimation methodology, we briefly summarize the Delft workbench and the H.264 video encoding application.

2.1. Architecture: Delft Workbench

The Delft Workbench is based on Molen Programming Paradigm. The Molen Paradigm is used to speedup an applications 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 [4].

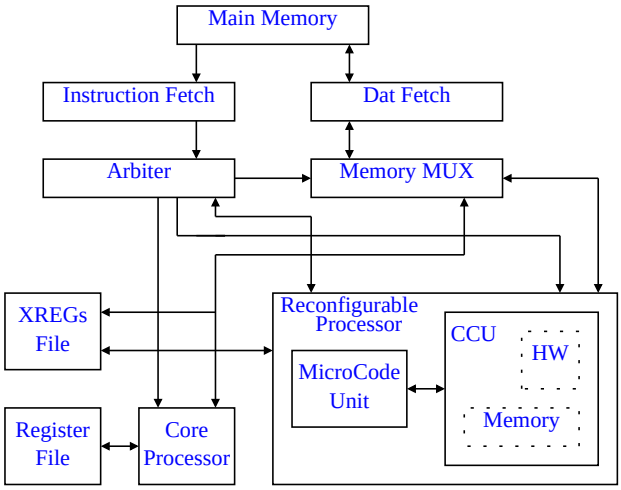

Figure 1. The Molen machine organization

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.

2.2. Application: H.264 Video Encoding

Block diagram of H.264 standard is shown in figure 2.

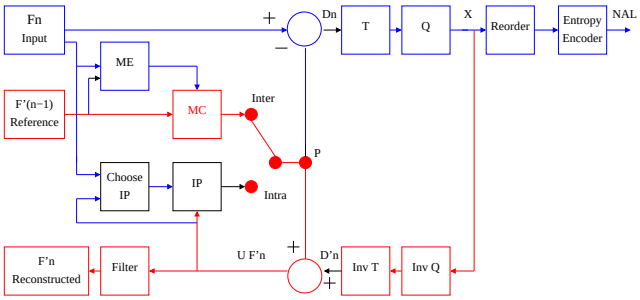

Figure 2. Block diagram of the H.264 encoder

An input frame *Fn* is presented for encoding. The frame is processed in units of a macroblock (corresponding to 16x16 pixels in the original image). Each macroblock is encoded in intra or inter mode. In either case, a prediction macroblock *P* is formed based on a reconstructed frame. In Intra prediction (IP) mode , *P* is formed from samples in the current frame n that have previously encoded, decoded and reconstructed. In Inter mode, *P* is formed by motion-compensated (ME and MC) prediction from one or more reference frame(s). The prediction *P* is subtracted from the current macroblock to produce a residual or difference macroblock *Dn*. This 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 macroblock (such as the macroblock prediction mode, quantizer step size, motion vector information etc) form the compressed bit-stream. This is passed to a Network Abstraction Layer (NAL) for transmission or storage.

3. PERFORMANCE ESTIMATION METHODOLOGY

The main steps in our performance estimation methodology for video encoding applications are shown in figure 3. The details of each step is described in [6]

3.1. Decomposition of Source Code

The source code (.c source files) of a video encoding application is analyzed. The objective of the application analysis is

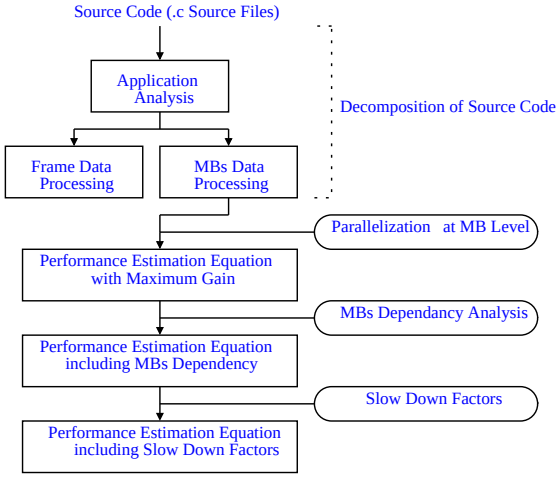

Figure 3. Proposed performance estimation methodology

to divide the source code into two types of processing parts: the frame data processing part and the macroblock (MB) data processing part. The example of the frame data processing is the parsing of video bit-stream into MBs. The example of the MB processing is the motion estimation (ME) step in a video encoder.

3.2. Parallelization at MB Level

The macroblock processing part is parallelized between different processing elements of the target heterogeneous reconfigurable architecture. The output of this step is the maximum possible performance gain with parallel execution (maximum parallelism).

3.3. Dependency Analysis of MBs

The performance gain obtained in second step is not true in reality because of the data dependency between MBs. There is a time duration during which maximum parallelism is not achieved. Dependency analysis is performed to formulate the sum of time duration before and after maximum parallelism.

3.4. Identification of Slow Down Factors

The performance gain obtained in third step is further reduced by some slow down factors for parallel execution. These factors may be, for example, the non uniform execution time and the idle time for reconfigurable processor.

4. H.264 ENCODER ANALYSIS

X264 [7] is an open source H.264 encoding algorithm. The profiling results of H.264 encoding with gprof [8] show 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

macroblock (MB Encode). However, these profiling results depend upon the frame size of the profiled video. If frame size is larger, the portion of analysis and encoding of macroblocks becomes higher.

If we summarize all of our analysis results, we can sketch the x264 encoder as shown in figure 4. The first block is *ReadFile* which parses the incoming video bit-stream into macroblocks. The processing in these blocks are at frame level. The next two block are *MB Analysis* and *MB Encode*. The processing in these blocks are at macroblock level. Again, at the end, the two blocks *CABAC* and *WriteFile* contain processing at frame level.

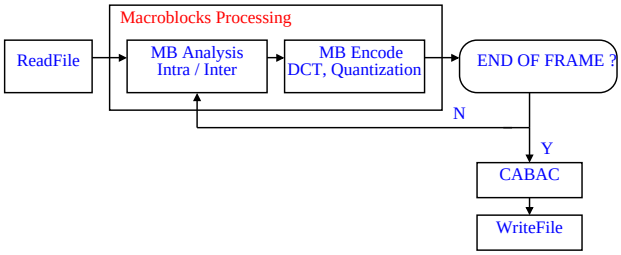

Figure 4. H.264 encoder division between frame data processing and macroblock data processing

5. HIGH LEVEL DESIGN FOR H264 ENCODER

On the basis of application analysis and performance estimation, we propose a high level design for H.264 encoder as shown in figure 5. The *video parsing* block takes input video and generates macroblocks (MBs).

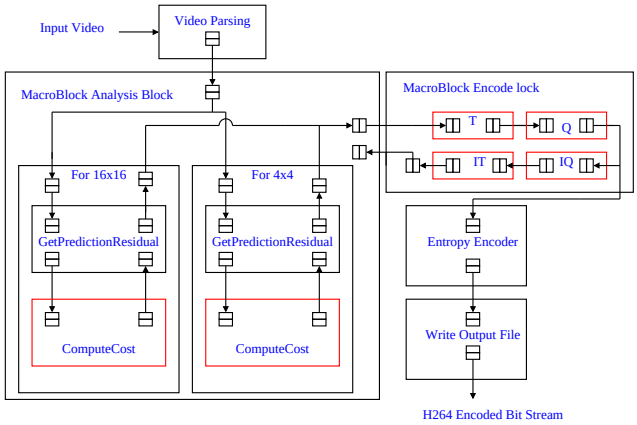

Figure 5. A high level design for H.264 encoder based on application analysis and performance estimation

The output of the *video parsing* block goes to the *MB Analysis block*. It knows the available boundary pixels based on the location of the MB in the frame. It starts MB Analysis (In the figure only two modes 16x16 and 4x4 prediction are shown for simplicity) on the current MB. It has two sub-blocks. One sub-block is for 4X4 prediction and other sub-block is for 16x16 prediction. Both of these sub-blocks are further subdivided into two modules. These modules are named as *GetPridictionResidual* and *ComputeCost*. The *GetPridictionResidual* calculates the predicted block and then the prediction residual, based on an input mode. The *ComputeCost* module uses this prediction residual to calculate the cost of that prediction mode. Finally both sub-blocks (one for 4X4 prediction and other for 16x16 prediction) returns their best residual to *MB Analysis* block. Then *MB Analysis* block picks the prediction mode and sends appropriate prediction residual through *Discrete Cosine Transform* (shown as T in figure 5) and *quantization* blocks (shown as Q in figure 5) for eventual *entropy encoding*, and backs through the inverse path to use for predicting the next MB.

5.1. Control and Data Parts

The functionality of *MB Analysis* block can be divided into the control and the data parts. The control part is in charge of iterating through all the possible prediction methods to choose the best one. The data manipulation part includes creating the prediction block and computing the cost associated with that block.

5.2. Determination of Boundary Pixels

The main control for *MB Analysis* block is to keep track of where the current MB is located in the frame. This is particularly important for determining what pixels are available for prediction. The input to the *MB Analysis* block is a single MB, so it must store all the data involved with its processing. In addition to keeping track of the location, the *MB analysis* block must also have all of the necessary boundary pixels from reconstructed MBs available. The left boundary of the current MB is taken from the MB that just finished the prediction process, so its pixels can easily be saved in a 16-entry register. The boundary pixels above the current MB, however, were predicted several macroblocks ago. Therefore, a register file of 16-entry vectors is required to store all of the boundary pixels needed by MBs later in the frame. The size of the register file is directly proportional to the width of the frame.

5.3. Generation of Residual

Once the *MB Analysis block* takes an input MB, determines which boundary pixels are available, and determines which pixels to extract from the register file, it sends a request to the 16x16 and 4x4 predictor via their input FIFOs and waits for their response. These blocks respond with their best prediction mode(s) and the associated cost from using that mode. After both sub blocks have this information ready, the MB

Analysis block makes a mode decision. The residual is then sent through the reconstruction chain for output and reconstruction. When the reconstructed data comes back to the *MB Analysis* block, the pixels that will be used for future prediction are saved in the register file and *MB Analysis* block is ready to process a new MB.

6. RELATED WORK

There are several approaches to parallelize the video encoding algorithms. The most popular choices are based on the GOP (group of pictures) level [9] , frame or slice level [10], combination of GOP and frame level [11] or at motion estimation level [12].

GOP level [9] encodes simultaneously several groups of consecutive frames. Frame level [10] encodes several slices of one frame in parallel. GOP parallelism gives good speed-up but imposes very high latency, on the other side frame parallelism gets less efficiency but low latency. Combining both approaches a compromise between speed-up and latency and then a broader spectrum of applications can be covered [11].

Parallelization at the frame level, however, does not fit for an heterogeneous platform like Delft workbench due to the space limitation of the Exchange Registers. Also dynamic allocation of data buffer in the Reconfigurable Processor of Delft Workbench is needed if the frame size is changed, which is not a good programming model for our target platform. Parallel execution at motion estimation level is the most popular technique for hardware implementation. If we use the same parallel execution on the Delft Workbench, we have to pay huge overhead of data transfer between the Reconfigurable Processor and the main memory. Therefore, we parallelize the algorithm at the MB level. Since the MB size is constant (16x16 for luminance and 8x8 for chrominance), we can allocate the data buffers statically. Hence, the exchange registers of a Reconfigurable Processor can accommodate all the required data and the code.

7. CONCLUSIONS

In this paper, we have proposed a performance estimation methodology for H.264 video encoding algorithm on the Delft workbench. Considering the communication architecture of the Delft workbench, we parallelized the H.264 encoding algorithm at macroblock level. On the basis of application analysis and performance estimation, a high level design for H.264 encoder was proposed.

Acknowledgments.

This work is sponsored by hArtes Project(IST-035143) supported by the sixth Framework Programme of the European Community under the thematic area "Embedded Systems". .

REFERENCES

[1] K Sigdel, R. J. Meeuws, K.L.M. Bertels, "A Profiling Framework for Design Space Exploration in Heterogeneous System Context." In *Proceeding of Prorisc Conference*, Veldhoven, The Netherlands, November 2007.

[2] H. J. Stolberg, S. Moch, L. Friebe, A. Dehnhardt, M. Kulaczewski, M. Berekovic, and P. Pirsch, "An Soc with two Multimedia DSPs and a RISC Core for Video Compression Applications." In *ISSCC.*, 2004, pp. 330–531.

[3] "ITU-T Rec. H.264 — ISO/IEC 14496-10 AVC, Document JVT-D157", Austria, July 2002.

[4] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K.L.M. Bertels, G.K. Kuzmanov, E. Moscu Panainte, "The Molen Polymorphic Processor.", In *IEEE Transactions on Computers*, 2004, Vol. 53, pp. 1363–1375...

[5] "International Standard Organization, Information Technology, Part2—Visual, ISO/IEC 14496-2".

[6] Muhammad Rashid, Jean-Christophe Le-Lann and Koen Bertels. "Video Encoding analysis for Parallel Execution on Reconfigurable Architectures" In *Summer Computer Simulation Conference: DASD Track*, Edinburgh, UK, 2008

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

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

[9] J.C. Fernndez and M. P. Malumbres, "A Parallel Implementation of H.264 Video Encoder.", In *Proceedings of EuroPar 2002*, Padderborn, pp. 830–833.

[10] A. Rodriguez, A. Gonzlez and M.P. Malumbres, "Performance Evaluation of Parallel MPEG-4 Video Coding Algorithms on Clusters of Workstations.", In *Conference on Parallel Computing in Elect. Engg.*, 2004.

[11] A. Rodriguez, A. Gonzalez, M.P. Malumbres, "Hierarchical Parallelization of an H.264/AVC Video Encoder.", In *Proceedings of the PARELEC06*, pp. 363-368

[12] Chuan.Y.C, Shiang-Y., J. Shung Wang, "An Embedded Merging Scheme for H.264/AVC Motion Estimation.", In *International Conf. on Image Processing*, 2003.