Examensarbete

H.264 Baseline Real-time High Definition Encoder on CELL

Examensarbete utfört i Datorteknik
vid Tekniska högskolan i Linköping
av

Zhengzhe Wei

LITH-ISY-EX--10/4355--SE

Linköping 2010
H.264 Baseline Real-time High Definition Encoder on CELL

Examensarbete utfört i Datorteknik
vid Tekniska högskolan i Linköping
av

Zhengzhe Wei

LITH-ISY-EX--10/4355--SE

Supervisor:  Di Wu
ISY, Linköpings Universitet

Examinator:  Dake Liu
ISY, Linköpings Universitet

Linköping, January 19, 2010
In this thesis a H.264 baseline high definition encoder is implemented on CELL processor. The target video sequence is YUV420 1080p at 30 frames per second in our encoder. To meet real-time requirements, a system architecture which reduces DMA requests is designed for large memory accessing. Several key computing kernels: Intra frame encoding, motion estimation searching and entropy coding are designed and ported to CELL processor units. A main challenge is to find a good tradeoff between DMA latency and processing time. The limited 256K bytes on-chip memory of SPE has to be organized efficiently in SIMD way. CA VLC is performed in non-real-time on the PPE.

The experimental results show that our encoder is able to encode I frame in high quality and encode common 1080p video sequences in real-time. With the using of five SPEs and 63KB executable code size, 20.72M cycles are needed to encode one P frame partitions for one SPE. The average PSNR of P frames increases a maximum of 1.52%. In the case of fast speed video sequence, 64x64 search range gets better frame qualities than 16x16 search range and increases only less than two times computing cycles of 16x16. Our results also demonstrate that more potential power of the CELL processor can be utilized in multimedia computing.

The H.264 main profile will be implemented in future phases of this encoder project. Since the platform we use is IBM Full-System Simulator, DMA performance in a real CELL processor is an interesting issue. Real-time entropy coding is another challenge to CELL.

Keywords
Video coding, H.264, CELL processor, Real-time coding, Intra prediction, Parallel programming, SIMD
Abstract

In this thesis a H.264 baseline high definition encoder is implemented on CELL processor. The target video sequence is YUV420 1080p at 30 frames per second in our encoder. To meet real-time requirements, a system architecture which reduces DMA requests is designed for large memory accessing. Several key computing kernels: Intra frame encoding, motion estimation searching and entropy coding are designed and ported to CELL processor units. A main challenge is to find a good tradeoff between DMA latency and processing time. The limited 256K bytes on-chip memory of SPE has to be organized efficiently in SIMD way. CAVLC is performed in non-real-time on the PPE.

The experimental results show that our encoder is able to encode I frame in high quality and encode common 1080p video sequences in real-time. With the using of five SPEs and 63KB executable code size, 20.72M cycles are needed to encode one P frame partitions for one SPE. The average PSNR of P frames increases a maximum of 1.52%. In the case of fast speed video sequence, 64x64 search range gets better frame qualities than 16x16 search range and increases only less than two times computing cycles of 16x16. Our results also demonstrate that more potential power of the CELL processor can be utilized in multimedia computing.

The H.264 main profile will be implemented in future phases of this encoder project. Since the platform we use is IBM Full-System Simulator, DMA performance in a real CELL processor is an interesting issue. Real-time entropy coding is another challenge to CELL.
First and foremost I would like to thank my supervisor Dr. Di Wu and examiner Prof. Dake Liu from Linköping University for showing me the path to H.264 encoder and giving me the possibility to complete this thesis. Thanks for your instruction and teaching hand by hand in all the time of this thesis work. Gratefully thanks to Prof. Liu for valuable comments and discussions on topics in real-time processing and thesis writing.

Not only technical problems, but homesickness, I’ve got many help from my former colleagues in the Division of Computer Engineering, ISY. I appreciate the time we stay together during the past one year. Yulin Huang, Jian Wang, Ningning Guo, and other friends in Linköping, make me feel happy and passionate to work and life.

Some friends from developerWorks forum supported me in my CELL research work. I want to thank them for sharing knowledge and experience with me.

Especially, I would like to give my deepest love to my parents in China. Without their patient love and encouragement, I couldn’t go ahead with this work.

Zhengzhe Wei
2010/01/19
# Table of Contents

## Chapter 1  Problem Statement

1.1 Motivation .......................................................................................... 1
1.2 Thesis Outline .................................................................................. 3

## Chapter 2  Background

2.1 Video Introduction ........................................................................... 5
2.2 Color Spaces .................................................................................... 6
2.3 Video Coding .................................................................................. 7
   2.3.1 Predictive Coding ....................................................................... 7
   2.3.2 Transform Coding ....................................................................... 8
   2.3.3 Entropy Coding .......................................................................... 8

## Chapter 3  H.264/AVC

3.1 Intra Prediction ................................................................................ 10
3.2 Inter Prediction ................................................................................ 12
3.3 Transform Coding ............................................................................ 12
   3.3.1 Transform .................................................................................. 12
   3.3.2 Quantization ............................................................................. 14
3.4 Entropy Coding ................................................................................. 14
   3.4.1 CAVLC ...................................................................................... 14
   3.4.2 CABAC ...................................................................................... 15
3.5 Debloking Filter ................................................................................. 16
Chapter 4  The CELL Processor ................................................................. 17
  4.1  CELL Architecture ........................................................................ 17
  4.2  Communications .......................................................................... 20
    4.2.1  Bus and I/O ........................................................................ 20
    4.2.2  Direct Memory Access ......................................................... 21
    4.2.3  Mailboxes and Signals ......................................................... 22

Chapter 5  Design and Implementation .................................................... 25
  5.1  Baseline Profile .......................................................................... 26
  5.2  System Architecture ..................................................................... 27
  5.3  Frame Partition and Scheduling .................................................... 29
    5.3.1  PPE Frame Reading .............................................................. 29
    5.3.2  SPE Frame Partition ............................................................ 31
  5.4  Communication and DMA ........................................................... 33
  5.5  SIMD implementation of I-frame Encoding ..................................... 37
    5.5.1  Vertical Prediction ............................................................... 38
    5.5.2  Horizontal Prediction ............................................................ 38
    5.5.3  DC Prediction .................................................................... 38
    5.5.4  Plane Prediction ................................................................. 39
  5.6  Motion Estimation ....................................................................... 41
    5.6.1  Reference Management ....................................................... 41
    5.6.2  Motion Estimation Algorithm .............................................. 42
  5.7  CAVLC ...................................................................................... 43

Chapter 6  Benchmark and Analysis .......................................................... 45
  6.1  Benchmark Configures ................................................................... 45
  6.2  Frame Performance ...................................................................... 46
  6.3  Cycle Performance ...................................................................... 50
  6.4  DMA Issues ................................................................................ 52
Chapter 7  Conclusion and Future Work..........................................................................................53
  7.1  Conclusion................................................................................................................................53
  7.2  Future work .................................................................................................................................54

REFERENCES.....................................................................................................................................55
List of Figures

Figure 2.1 Overview of one simple video coding system ................................................................. 7
Figure 3.1 Generalized block diagram of an H.264 encoder .......................................................... 10
Figure 3.2 Intra 4 x 4 prediction mode directions .............................................................................. 11
Figure 3.3 Schematic Block Diagram for CABAC ............................................................................ 15
Figure 4.1 A general architecture of CBE ......................................................................................... 18
Figure 4.2 PowerPC Processor Element (PPE) basic block diagram ................................................. 18
Figure 4.3 SPE architectural block diagram .................................................................................... 19
Figure 5.1 General architecture of PPU and SPU on CELL .............................................................. 28
Figure 5.2 One 6x4 frame in YUV420 format .................................................................................. 30
Figure 5.3 Horizontal encode frame partitions for SPUs ................................................................. 31
Figure 5.4 Vertical encode frame partitions for SPUs ....................................................................... 32
Figure 5.5 Vertical reference frame partitions for SPUs ................................................................. 34
Figure 5.6 Flow chart of PPU-SPU double buffering DMA example .............................................. 37
Figure 5.7 Plane prediction computation using SIMD structure ..................................................... 40
Figure 6.1 PSNR per frame in three videos ....................................................................................... 47
Figure 6.2 I frame of video sequence Sunflower ............................................................................. 48
Figure 6.3 The motion vectors around bee body in Sunflower Video ............................................. 50
List of Tables

Table 5.1  Normal application requirements for H.264 profiles ................................................................. 26
Table 5.2  PPU command list from SPU's sight. ............................................................................................. 33
Table 5.3  CAVLC parameters ..................................................................................................................... 43
Table 6.1  The average bit size for one encoded frame. Metric: K bits ......................................................... 49
Table 6.2  Two types Intra macroblocks for three videos. Metric: macroblocks(16x16) ............................... 49
Table 6.3  The average p-frame PSNR in three videos. Metric: dB ............................................................. 49
Table 6.4  Performance matrix of Intra mode detection and SAE calculation ............................................. 51
Table 6.5  Cycle usage for one macroblock partition (1632 macroblocks per partition) ............................. 51
Chapter 1  Problem Statement

The H.264/MPEG-4 AVC is the newest international standard completed in May 2003 for video coding. High compression performance and network friendly are the main goals of H.264/AVC. With the development of Blu-ray Disc, HDTV and high quality videos from the Internet, H.264 is most popular video coding standard nowadays [1]. Compared to previous video coding standards, H.264 complexity is about four times that of MPEG-2 and two times that of MPEG-4 Visual Simple Profile [2].

1.1 Motivation

High compression performance leads to high computational complexity and high CPU utility. Real time high definition video encoding is needed in industries like living HDTV news. The coding computation becomes more complex significantly and most general CPUs are insufficient in computation powers dealing with HD contents [1]. To do the real time encoding and to offload the CPU, parallelized computing is applied. With eight powerful SIMD processing units managed by one PowerPC core, the STI CELL Broadband Engine processor is an ideal platform for the H.264 parallelization [3]. Comparing with hardware encoders, CELL processor can provide general computations and flexible extensions.
Lots of works have been done in low definition or non-real time encoding since the occurrence of the first draft of H.264. But real time HD H.264 parallel processing is totally new and full of challenges. The motivation for this master thesis is to implement an encoder in CELL platform and evaluate the maximum computation ability of the CELL processor. Lim Boon Shyang and Di Wu researched the state of art for CELL HD real time video encoding (1080p non-interlaced @ 30 fps) [4]. Under their well-developed architecture, only P frame encoding with a simple motion estimation algorithm was implemented.

As the continuation of Lim, we want to develop a complete encoder so that we can evaluate more accurate performance. We select the baseline profile of the H.264 standard as our implementation profile. Basically, there are three parts of work to approach the baseline profile.

I- I-frame encoding is the basic requirement in the H.264 baseline profile. I-frames are encoded independently in the video sequence to have random access capability and simple fast forwarding. The first goal of our project is the implementation of I-frame encoding.

II- Motion estimation search range is limited to 16 in Lim’s work [4], which only utilized a fraction of 51.8% processor computation power in average. To meet encoding requirements of high speed video sequences, we will enhance the motion estimation algorithm performance and increase the motion estimation search range up to 64. The finding of different search ranges will be used to design further main or higher profiles for H.264 real time HD encoding. This is the second goal in the project. Note that data needed by 64x64 search ranges will extend the memory limitation of CELL processing units. We have to design the new instruction level memory access architecture as our third project goal.

III- Entropy coding is designed and implemented in the baseline profile to have a basic evaluation as the additional objective we interest. Since the limitation of CELL and the tree algorithm, entropy coding can only be implemented in non-real-time at this moment.

The project is based on IBM CELL Software Development Kit version 3.1 (SDK 3.1) and IBM Full-System Simulator version 3.0 for the Cell Broadband Engine Processor.
1.2 Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2, the basic concepts and theories used in this work are described. Chapter 3 introduces H.264 encoding techniques related in this area. It’s assumed that the reader has a good understanding of H.264. Chapter 4 is concerned with the general architecture of the CELL processor. Readers who are familiar with CELL can skip this chapter. Chapter 5 presents the designs and algorithms used in the implementation. In Chapter 6, we describe the experimental results including a comparison of the previous work. Chapter 7 concludes the thesis and points out future work.
In this chapter, we introduce the background in the area of digital video encoding, and to describe the general components in the video coding system and coding techniques used throughout this thesis.

2.1 Video Introduction

Video is a time sequence of images. Generally it refers to the technology of capturing, recording, processing, storing, transmitting, and reconstructing the time sequence of still images representing scenes. Digital video comprises a series of two-dimensional (2-D) bitmap digital images at a constant rate. The 2-D image is projected from a dynamic three-dimensional (3-D) scene onto a digital video camera or other equipments. One 2-D still digital image is called a frame of the video. It consists of a two dimensional array of pixel values with color information. The number of frames displayed per second is a main feature of videos. We measure this frame rate as frames per second (FPS). The frame size is WxH where W is the video width of pixels and H is the height of pixels. Pixels have only one property, color information, to describe them. The color of a pixel is represented by a fixed amount of bits. The more bits the more subtle variations of colors the pixel can reproduce. This is called the color depth of the pixel, which is the constant value in one video.
An example uncompressed video sequence has 1 hour (3600s) duration. The frame size of this example video is 1920x1080 (WxH) at a color depth of 24bits and video frame rate is 30fps. This video has the following properties:

- **pixels per frame** = 1920 * 1080 = 2,073,600 pixels
- **bits per frame** = 2,073,600 * 24 = 49,766,400 bits = 47.46 Mbits
- **bit rate** = 47.46 * 30 = 1423.8 Mbits/sec (1423.8 Mbps)
- **video size** = 1423.8 * 3600 = 5,125,680 Mbits = 5005.5 Gbytes

### 2.2 Color Spaces

There are two systems of color space using in the video and image. The RGB color space uses three basic colors Red, Green and Blue to represent each sample or pixel. It is common for image encoding and presentation. In the RGB color space one pixel needs three different R/G/B values to store. None of these three values can be ignored. This kind of equal values makes it difficult to obtain compression in videos.

Since the Human Visual System (HVS) is more sensitive to luminance (brightness) than to chrominance (color), a better way to represent an image is to store the luminance information, Y, with higher resolution than the chrominance information, Cb, Cr and Cg. This is YUV color space.

The luminance value is calculated by weighted factors from R, G and B values.

\[ Y = k_r R + k_g G + k_b B \]

where \( k_r, k_g, k_b \) are weighting factors. \( k_r = 0.299, \ k_g = 0.587 \) and \( k_b = 0.114 \) are often used in normal processing.

The color values are calculated as the difference between RGB and the luminance Y:

- \( C_r = 0.713(R - Y) \)
- \( C_b = 0.564(B - Y) \)
- \( C_g = G - Y \) (redundant value, abandoned)

The YUV 4:2:0 sampling format is the most common and is used in DVD and digital television. In 4:2:0 the chroma components, Cr and Cb, have half the resolution compared to the luma component, Y. We will discuss the format of YUV420 in Chapter 5.
2.3 Video Coding

Codec stands for Coder/Decoder. With video codec, the system codes the digital video data and recognizes the format built by the specified codec for playbacks. In our project, a completed coder part of H.264/MPEG-4 AVC codec on CELL processor is the final objective. The most important technique used in the coder part is video coding.

![Overview of one simple video coding system.](image)

Video coding is the space saving representation of video data, for storage or transmission, for both analog and digital video. The biggest challenge is to reduce the size of the video data using video compression. Video compression is the compression of a sequence of digital images. The quality and size of encoded video data must be considered for different video coding methods. Three key methods will be introduced in following sections. The general system architecture of the video coding is illustrated in Figure 2.1.

2.3.1 Predictive Coding

The predictive way of using encoded information in video data to determine current encoding is called predictive coding. Normally symbols or picture partitions in the frame have very high probability that they are correlated with adjacent symbols or partitions in the same frame. We can save only the difference between adjacent encoded zones and current encoding part to reduce the redundancy inside one frame. This kind of predictive coding method is named intra-prediction.
In the video sequence to be encoded, certain parts of the same symbols or pixels occur repeatedly in consecutive frames only with very small changes. If the coding is based on other pictures in the sequence, which is available to both encoder and decoder, then there might not be any need to transit the additional information rather than the small difference or changing between frames. We call this small difference between frames as prediction error value, and this predictive manner to be inter-prediction.

2.3.2 Transform Coding

The transform coding is a technique that decomposes or transforms the video source into encoded data according to the desirable characteristics. Discrete Cosine Transform (DCT) is a popular transform in the signal and image processing. By compacting most of the information in a source sequence into a few elements of the transformed sequence using a reversible transform, and then discarding the elements of the sequence that do not contain much information, we can get a large amount of compression.

In the video frequency domain, high or very high frequencies are not very sensitive to humans. Like the sound frequency limitations, human eyes only accept a certain range of video frequencies. So the high frequency part can be removed without decreasing the visual quality of the video.

2.3.3 Entropy Coding

According to Shannon's source coding theorem, the optimal code length for a symbol is \(-\log_b P\), where \(b\) is the number of symbols used to generate output codes and \(P\) is the probability of the input symbol [1]. Entropy is a measure of symbol similarity between video stream data. We can use entropy coding to reduce the code length for a symbol occurred in the video stream context. Normally the most common symbols use the shortest codes. Two most common entropy encoding techniques are Huffman coding and arithmetic coding.
The Joint Video Team of ISO/IEC Moving Picture Experts Group and ITU-T Video Coding Experts Group finalized the draft new standard for the coding of natural video images in March 2003. The new standard [1] was known as H.264 and also MPEG-4 Part 10, “Advanced Video Coding” for formal approval submission as H.264/AVC.

The goals of H.264/AVC were enhanced video compression efficiency and network friendly video representation for both interactive and non-interactive applications. Compared with previous standards, H.264 achieves at least 50% improvement in bit-rate efficiency [2]. It has been adopted by many application standards such as Blue-ray and HD-DTV.

Only the syntax and semantics of the bitstream and the decoding process are standardized in the standard. To the encoding process, it gives great flexibilities to the encoder for best encodings on the requirements of the application. The encoded video generated by the encoder must be decoded according to the H.264 standard. We will describe several essential processes inside the encoder as illustrated in Figure 3.1.

More H.264 introductions can be found in [2] [6].
3.1 Intra Prediction

Intra prediction of a macroblock is that this macroblock is encoded using only already encoded and reconstructed neighboring macroblocks of the same image without reference to other frames in the video sequence. It provides access points to the coded sequence where decoding can start. In H.264/AVC, an intra macroblock has its luminance component and chrominance components separately predicted. Now H.264 supports different types of intra coding, Intra_4x4, Intra_8x8 or Intra_16x16 for the prediction of the luminance component Y and Intra_8x8 for chrominance components Cb and Cr. Intra_8x8 is only implemented in High profile or higher ones. Intra_16x16 is used when the macroblock is smooth and it can be divided into 16 Intra_4x4 sub-blocks when the macroblock contains detailed information. I_PCM is a special intra coding type for a macroblock that transmits the values of the image samples directly without prediction or transformation.

H.264 uses the encoded and reconstructed blocks (before deblocking) to form different prediction blocks for current encoding block and selects the prediction block with the minimum error.
value to the original block. The residual signal or error value between the original block and the prediction is finally encoded in the next processing. One prediction case is selected from a total of 9 prediction modes for each Intra_4x4 and Intra_8x8 luma blocks; 4 modes for each Intra_16x16 luma block; and 4 modes for each chroma blocks.

Figure 3.2 shows a 4x4 luma block that is to be predicted as Intra_4x4. For the current block of 4x4 predicted samples [a, b, ..., p], the above and left previously reconstructed samples [A, B, ..., M] are used for prediction blocks generation according to direction modes. The arrows in Figure 3.2 indicate the direction of prediction of pixels in each mode. For mode 0 (vertical), the predicted block are formed by extrapolation from upper samples [A, B, C, D]. For mode 1 (horizontal), the predicted block are formed by extrapolation from left samples [I, J, K, L]. For mode 2 (DC), all of the predicted samples uses the same value that is the mean of upper and left samples [A, B, ..., K, L]. For mode 3 (diagonal down left), mode 4 (diagonal down right), mode 5 (vertical right), mode 6 (horizontal down), mode 7 (vertical left), and mode 8 (horizontal up), the predicted samples are formed from a weighted average of the prediction samples A-M that are defined in 8.3.1.1 of [6]. Before applying predictions, all modes have to check if all of the required prediction samples are available.

![Figure 3.2 Intra 4 x 4 prediction mode directions](image)

(vertical -- 0; horizontal -- 1; DC -- 2; diagonal down left -- 3; diagonal down right -- 4; vertical right -- 5; horizontal down -- 6; vertical left -- 7; horizontal up -- 8)

For Intra_8x8, one mode is selected from the 9 similar modes to the Intra_4x4 prediction. For Intra_16x16, four modes are available. For mode 0 (vertical), mode 1 (horizontal), mode 2 (DC), the predictions are similar with the Intra4x4. For mode 4 (Plane), a linear plane function defined in 8.3.3.4 [6] is fitted to upper and left neighboring pixels.

Each chroma component of a macroblock is predicted from chroma samples above and/or to
the left that have previously been encoded and reconstructed. The chroma prediction is defined for one block size, Intra_8x8 chroma prediction. 4 prediction modes for Intra_8x8 chroma prediction are very similar to the Intra_16x16 luma prediction modes, except that the order of mode numbers is different: mode 0 (DC), mode 1 (horizontal), mode 2 (vertical), and mode 3 (plane).

The Sum of Absolute Errors (SAE) for each prediction indicates the magnitude of the prediction error. The prediction mode with the smallest SAE will be selected as the best match block. This prediction mode information and SAE are transferred to next phases of H.264 encoding instead of the generated prediction block.

### 3.2 Inter Prediction

Inter prediction generates a prediction model from one or more previously encoded frames based on block-matching motion compensation. A major advantage in H.264 is that it supports a range of sub-block sizes of 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4 pixels motion estimation. Smaller sub-block sizes can find better matches in a high complexity video sequence but increases the motion vector length at the same time. Another major advantage in H.264 is sub-sample motion vectors. A new sub-pixel image created by interpolation between inter pixels will be estimated to find a better match. There are half-pixel motion estimation and quarter-pixel motion estimation used in H.264. The best motion vector will be compensated and encoded in integer pixel motion estimations and sub-pixel motion estimations.

The best matching block may be not only in the most previous encoded reference frame. Multiple reference frames can be stored in frame lists for a better compression search. Depending on the reference frames stored in the encoder and decoder, H.264 provides many options for choosing the best prediction references for macroblocks. P slices only use previous encoded frames. B slice may be predicted from reference frame before or after the current frame in temporal order. More memory and computational complexity are required in the encoder and decoder if multiple reference frames are used.

### 3.3 Transform Coding

#### 3.3.1 Transform

Each macroblock after the prediction is transformed, quantized and coded. Previous standards such as MPEG-2 and H.263 applied the 8x8 Discrete Cosine Transform (DCT) as the
basic transform. H.264 uses three transforms depending on the type of macroblock data that is to be coded: a Hadamard transform for the 4x4 array of luma DC coefficients in intra macroblocks (predicted in 16x16 mode), a Hadamard transform for the 2x2 array of chroma DC coefficients (in any macroblock) and a DC-based transform for all other 4x4 blocks in the residual data.

The 4x4 residual transform operates on 4x4 blocks of residual data after motion-compensated prediction or Intra prediction. The transform is based on the DCT but with some fundamental differences: 1. It is an integer transform that all operations can be carried out with integer arithmetic. 2. The inverse transform is fully specified in the H.264 standard and if this specification is followed correctly, mismatch between encoders and decoders should not occur. 3. The core part of the transform is multiply-free, only addition and shift operations are required. 4 a scaling multiplication (part of the complete transform) is integrated into the quantizer (reducing the total number of multiplications).

The entire process of transform and quantization can be carried out using 16-bit integer arithmetic and only a single multiply per coefficient, without any loss of accuracy.

The 4x4 DCT of an input array X can be denoted as:

\[ Y = (C X^T) \otimes E \]

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & d & -d & -1 \\
1 & -1 & -1 & 1 \\
d & -1 & 1 & -d
\end{bmatrix}
\begin{bmatrix}
X
\end{bmatrix}
\begin{bmatrix}
1 & 1 & 1 & d \\
1 & d & -1 & -1 \\
1 & -d & -1 & 1 \\
1 & -1 & 1 & -d
\end{bmatrix}
\begin{bmatrix}
a^2 & ab & a^2 & ab \\
ab & b^2 & ab & b^2 \\
a^2 & ab & a^2 & ab \\
ab & b^2 & ab & b^2
\end{bmatrix}
\]

\((C X^T)\) is a “core” 2-D transform. E is a matrix of scaling factors and the symbol \( \otimes \) indicates that each element of \((C X^T)\) is multiplied by the scaling factor in the same position in matrix E (scalar multiplication rather than matrix multiplication). Number a b c d are constants defined in the DCT.

To simplify the implementation of the transform, d is approximated by 0.5. So the refined forward transform can be rewritten as:

\[
Y = \begin{bmatrix}
1 & 1 & 1 & 1 \\
2 & 1 & -1 & -2 \\
1 & -1 & -1 & 1 \\
1 & -2 & 2 & -1
\end{bmatrix}
\begin{bmatrix}
X
\end{bmatrix}
\begin{bmatrix}
1 & 2 & 1 & 1 \\
1 & 1 & -1 & -2 \\
1 & -1 & -1 & 2 \\
1 & -2 & 1 & -1
\end{bmatrix}
\begin{bmatrix}
A^2 & AB/2 & A^2 & AB/2 \\
AB/2 & B^2 & AB/2 & B^2/4 \\
A^2 & AB/2 & A^2 & AB/2 \\
AB/2 & B^2/4 & AB/2 & B^2/4
\end{bmatrix}
\]

13
Where A and B are the approximate value of a and b.

### 3.3.2 Quantization

After the forward transform, most of the signal values are concentrated in a range of low frequencies. Then we can apply the quantization to scale down the coefficients and reduce signal space in another way. Different quantizer step size (qstep) can be selected by the encoder to meet application requirements. Normally the qstep value is from 0 to 51 by standard and these are indexed by a Quantization Parameter (QP). The value of qstep corresponds to each QP. The wide range of quantizer step sizes makes it possible for an encoder to accurately and flexibly control the trade-off between bit rate and quality.

### 3.4 Entropy Coding

The H.264 standard specifies two types of entropy coding: Context-based Adaptive Variable-Length Coding (VLC) and Context-based Adaptive Binary Arithmetic Coding (CABAC).

#### 3.4.1 CAVLC

The Variable-Length Coding scheme, part of the Baseline Profile of H.264, is described in this section. Since we only focus on the baseline profile, CAVLC is what we interest in and implement in the project.

After transform and quantization, the probability that the value of coefficients is zero or +/-1 is very high. CAVLC handles the zero and +/-1 coefficients as the different manner with the levels of coefficients. The total numbers of zero and +/-1 are coded. For other coefficients, their levels are coded.

1. After prediction, transformation and quantization, blocks are typically sparse (containing mostly zeros). CAVLC uses run-level coding to compactly represent strings of zeros.

2. The highest non-zero coefficients after the zig-zag scan are often sequences of +/-1. CAVLC signals the number of high-frequency +/-1 coefficient (“Trailing 1s” or “T1s”) in a compact way.
3. The number of non-zero coefficients in neighboring blocks is correlated. The number of coefficients is encoded using a look-up table; the choice of look-up table depends on the number of non-zero coefficients in neighboring blocks.

4. The level (magnitude) of non-zero coefficients tends to be higher at the start of the reordered array (near the DC coefficient) and lower towards the higher frequencies. CAVLC takes advantage of this by adapting the choice of VLC look-up table for the “level” parameter depending on recently-coded level magnitudes.

### 3.4.2 CABAC

CABAC is designed for significantly improved coding efficiency. It’s based on three key elements: binarization, context modeling, and binary arithmetic coding. Compared to CAVLC, CABAC can reduce the bit rate 5-15%. Interlaced TV signal has the highest performance improvement (15%) if CABAC is used. For each symbol element is encoded as shown in Figure 3.3.

![Schematic Block Diagram for CABAC](image)

**Figure 3.3** Schematic Block Diagram for CABAC

**Step 1** - binarization;

A given non binary valued symbol (e.g. a motion vector or transform coefficient) is uniquely mapped to a binary sequence. This process is similar to the process of converting a data symbol into a variable length code for further encoding with the arithmetic coder.

**Step 2** - context modeling;

A context probability model will be applied for one or more elements of the binarized symbol after Step 1). The probability model is selected depending on the associated cod-
ing-mode decision with previously encoded syntax elements.

Step 3) binary arithmetic coding;

An arithmetic coder encodes each element according to the selected probability model together with a subsequent model updating.

### 3.5 Deblocking Filter

The discontinuity may occur if block-based video coding is used. H.264 may suffer from this discontinuity due to block-based transform in intra and inter prediction coding, and the quantization of the transform coefficients. The deblocking filter can decrease the discontinuity in the block boundary and reduce the difference between the reconstructed block and the original block. So it prevents the propagation of accumulated coded noise in total.

In H.264, the deblocking filter works on one macroblock at a time. Filtering is applied to horizontal or vertical edges of 4x4 blocks within the macroblock. The luma deblocking filter process is performed on four 16-sample edges and the deblocking filter process for each chroma components is performed on two 8-sample edges. When the whole reconstructed frame is deblocking filtered, it will be stored in the encoded data stream and may be used as a reference frame for other frames of the video sequence.

The deblocking can be applied adaptively at several levels according to different standards. There are three deblocking levels in H.264/AVC.

- **Slice level:** applying the global filtering strength to the entire video slice.

- **Block-edge level:** modifying with the filtering strength made dependent on inter/intra prediction mode, motion differences, and the presence of coded residuals in two participating blocks.

- **Sample level:** filtering for each individual sample.
The purpose of this chapter is to introduce the basic concepts in STI Cell Broadband Processor Architecture (CBEA). Two function parts will be described respectively. The communications between different processor units, including Direct Memory Access (DMA) and Mailboxes, are our essential content in this chapter.

4.1 CELL Architecture

Based on the 64-bit PowerPC Architecture, the Cell Broadband Engine (CBE) or CELL is a new family of microprocessors developed through the collaboration of Sony, Toshiba and IBM (STI). It has been using in game consoles like PlayStation 3 and broadband content consumer-electronics devices such as high-definition televisions since early 2001. The goals of CELL architecture are highly parallel, very fast, with huge memory throughput [3].

The CBE is a single-chip multiprocessor with a 64-bit PowerPC Processor Element (PPE) and eight specialized independent coprocessors (Synergistic Processor Elements (SPEs)). It is a very powerful processor which is highly suitable for game and multimedia processing. The overview of CBE architecture is shown in Figure 4.1.
Chapter 4  The CELL Processor

Figure 4.1  A general architecture of CBE.

PPE is a 64-bit RISC PowerPC core that is compliant with existed 32-bit and 64-bit software designed for the 64-bit PowerPC architecture. The new advance in PPE is the Vector/SIMD multimedia Extension instruction set. As a normal main processor, PPE supports the operating system, manages system resources and controls SPE processing. We can see the two main units, PowerPC Processor Element (PPE) and PowerPC Processor Storage Subsystem (PPSS), in the PPE from Figure 4.2.

Figure 4.2  PowerPC Processor Element (PPE) basic block diagram.
The PPU supports two simultaneous threads of execution, which processes two tasks simultaneously and shares the dataflow between these two cores to software. Instruction control and execution is the work of PPU. Seven execution units are used to complete PPU’s work, instruction-control unit, load and store unit, fixed-pointed integer unit, floating-point unit, vector unit, branch unit and virtual-memory management unit. The collaboration between all units and Vector/SIMD Multimedia Extension associated C intrinsic extensions provides vast performance improvement to broadband content multithread allocations. The PPSS unit responses all memory requests to the PPE from the PPE itself, SPEs, and other I/O devices through EIB. PPSS includes a bus interface unit for EIB and a 512KB L2 instruction and data cache. Since memory requests have different schedules and priorities, PPSS has various queues to handles the requests. The L2 instruction and data cache uses replacement-management tables to control the program cache usage with error correcting.

The eight SPEs are more adept and optimized for compute-intensive tasks and data-rich operations. Each SPE contains a RISC core, 256K Byte Local Storage (LS) which may store program, stack, local data structures and DMA buffers controlled by software, and a 128-bit 128-entry unified register file. SPEs are designed to process data, so they are slower than the PPE at task switching. As shown in Figure 4.3, we can two main parts in the SPE, Synergistic Processor Unit (SPU) and the Memory Flow Controller (MFC). MFC will be described in details in the next section.

![SPE architectural block diagram.](image)
Chapter 4  The CELL Processor

The SPU is a private system-on-chip or processor with the processing unit connected to 256KB LS. It’s optimized to run SPE threads generated and managed by the PPE. The SPU uses instructions and data in its LS. The LS is unprotected and un-translated storage for the easy accessing among all SPUs. LS can be filled by requesting DMA transfers from SPU’s MFC part. The MFC provides communication interfaces between SPUs and the EIB. It also implements operation synchronization and bus bandwidth reservation when the SPE collaborates with other parts in the system.

In each SPU there are two pipelines, odd and even, that provides the ability to execute two instructions per cycle. Generally, the odd pipeline response memory load / store and memory permute and the even pipeline executes floating /fixed point instructions. Similar to the PPE Vector/SIMD Multimedia Extension instruction set, the instruction set for the SPEs is a new set of SIMD instructions, the Synergistic Processor Unit Instruction Set Architecture, with accompanying C/C++ intrinsic. The unique instruction set for DMA management, external events, inter-processor communication, and other functions is included as the additional part of the whole general set. The SPE SIMD engine provides 128-entry 128-bit SIMD general purpose register file and up to 16-way SIMD data parallelism. More detailed information about PPE and SPE can be found in reference [3].

4.2 Communications

Communications define the collaborated work between any two units in CELL processor. DMA and mailbox are main methods we use in the thesis.

4.2.1 Bus and I/O

The communications between all processing elements, memory/storage and the external I/O are provided by the Element Interconnect Bus (EIB). The EIB supports full memory-coherent and symmetric multiprocessor (SMP) operations. When a CELL cluster is applied, EIB buses are used to connect with each other coherently. The EIB consists of four 16-byte-wide data rings, each of which transfers 128 bytes data at a time. Two data rings are clockwise and the other two are counterclockwise. Processor elements can drive and receive data simultaneously through EIB. Multiple transfers can be processing concurrently on each ring, including more than 100 outstanding SPE DMA memory transfer requests. The internal bandwidth of the EIB is 96 bytes per clock cycle.

The EIB also connects to the Memory Interface Controller (MIC) to access the main storage and the Cell Broadband Engine Interface (BEI) to communicate with I/O devices. The MIC
access the physical memory with Rambus extreme data rate (XDR) I/O (XIO) memory interfaces. Bandwidths of each interface are 1 to 8, 16, 32, 64, or 128 bytes with coherent memory ordering. Up to 64 read requests and 64 write requests can be queued. The BEI provides I/O interfacing including address translation, command processing, interrupt controller, and bus interface controller. The BEI supports two Rambus FlexIO interfaces. One interface uses only a noncoherent I/O interface (IOIF) protocol, a kind of I/O devices suitable protocol. The other interface is software selectable between the noncoherent IOIF protocol and the memory-coherent Cell Broadband Engine interface (BIF) protocol, which is the EIB’s internal protocol. The BIF protocol can be used to extend the EIB to another memory-coherent device. The CELL cluster uses the BIF protocol as its communication protocol between different individual cell processors through the BEI.

4.2.2 Direct Memory Access

Since SPEs are the main units of data computing and SPEs fetch instructions from their own LS, a large part of work in one program is loading main storage or physical memory data into SPEs’ LS. The SPEs support a special SIMD instruction set (Synergistic Processor Unit Instruction Set Architecture) and a unique set of commands for managing DMA transfers and system messaging.

As we introduced in section 4.1, DMA transfers and other communications between the SPE and the system are implemented and managed by the SPE’s Synergistic Memory Flow Controller (MFC). After the SPE initiates the DMA transfer by issuing a DMA command, the MFC queues the DMA command into the right queue and maintains and processes different queues. The working SPE doesn’t have to wait for the DMA completion so it can continue to execute following instructions or operates on previously transferred data. At the same time, MFC selects one DMA queue and processes the DMA command asynchronously and autonomously.

The MFC has two ways to process DMA commands, individual processing and list processing (such as a scatter-gather list). The MFC can execute a sequence of DMA transfers that is constructed in an SPE’s local store if the DMA-list processing command is given. The maximum data size of one DMA is 16KB. And the DMA-list command can call up to 2048 DMA transfers in one list, 32768KB data in total. This approach to accessing memory can schedule DMA transfers effectively to hide memory latency and lead to high application performance, especially in the discontinuous main storage accessing.

The DMA command is issued by specifying a pair of a local address in LS and a remote address in main storage. DMA transfers access main storage using PowerPC effective addresses. As in the PPE, SPE address translation is managed by PowerPC Architecture segment and page tables, which are loaded into the SPEs by privileged software running on the PPE. The SPEs
are not intended to run an operating system.

The MFC channels are unidirectional message-passing interfaces. The PPE and other devices in the system, including other SPEs, can also access this MFC state through the MFC’s memory-mapped I/O (MMIO) registers and queues, which are visible to software in the main-storage address space.

### 4.2.3 Mailboxes and Signals

The mailbox is one kind of inter-process or inter-unit communication mechanisms in the operating system. It uses queues for data exchanging to pass the control or the content information. In CELL, mailbox, which is implemented in MMIO registers of each SPE’s MFC part, is primarily intended for communications between an SPE and other devices. Each mailbox has an SPE channel assignment as well as a corresponding MMIO register. The SPE side uses read or write channel instructions to access the mailbox. And the PPE and other SPEs use load and store instructions to access the corresponding MMIO addresses.

There are two different mailboxes, privileged-state and problem-state, refers to the SPU Write Outbound Mailbox and the SPU Outbound Interrupt Mailbox respectively for sending messages out from an SPE. Only one mailbox, the SPU Read Inbound Mailbox, is provided for receiving messages to the unit. Mailbox is a blocking operation. A write operation to an outbound full mailbox stalls the SPE until a free slot is created in the mailbox after a PPE read operation. Similarly, a SPE read from an empty inbound mailbox is stalled until the PPE or an SPE writes to the mailbox.

SPU signal-notification channels are inbound (to an SPE) 32-bit registers. The PPE or other units can read or write the corresponding MMIO register. Only the local signal-notification channels can be accessed inside SPEs. Two channels can be configured, one-to-one signaling is in the overwrite mode that overwrites previous content of the MMIO address if a signal comes, and many-to-one signaling adds the new 1 bits into the current content. An SPE read of one of its two signal-notification channels clears the channel atomically. On the contrary, a PPE MMIO read does not clear the channel. An SPE read from the signaling channel will be stalled when no signal is pending at the time of the read [3]. One-to-one signaling has the same results as the using of mailboxes.

Since the data are divided into several different SPEs for computing, we need the SPE preparation to start DMA and the data synchronization to make sure that all data are processed and ready for the next computing. The PPE, which is the application task controller, will notifies the right SPE by either writing to the SPU Mailbox or writing to one of the SPE’s signal notification registers. When the computation is complete in the SPE, SPU Write Outbound Mail-
box will be used to notify the PPE for further processing. If needed, the PPE can send back the acknowledgement mailbox or signal to control the SPE to process result-related data.
In this chapter the system architecture of design and implementation are proposed. The proposed design uses the existed architecture as its communication system and most units of the implementation are done on the SPE part. Fundamental structures and functions are defined based the architecture.

The H.264 baseline profile, which defines our boundary of work, is introduced first. Then the overview of system architecture is ported on PPE and SPEs. In the next section, frame reading and partitioning are designed for job scheduling and DMA efficiency. Communication commands and DMA issues are designed too. Intra prediction is one of essential modules in our encoder and it will be implemented in SIMD intrinsics. The last part gives a refined hexagon search algorithm that includes a computational complexity analysis.

Non-interlaced YUV4:2:0 1920×1080 pixels resolution video sequence at 30 frames per second is the target source in our design. Only luminance component Y is read and encoded in this version of the real-time encoder.


5.1 Baseline Profile

In H.264/AVC, 6 profiles are defined, namely baseline profile, main profile, extended profile, high profile, high 10 profile and high 4:2:2 profile [6]. Different profiles define a set of capabilities or functions that a coding standard may use. For each profile, 15 levels are specified in upper bound or lower bound perimeters, e.g., video bit rate and motion estimation search range. These profiles serve a wide range of applications, bit rates, resolutions, qualities, and services. There is no strong relation between profiles and applications. We can see the normal profiles suing in different applications in Table 5.1.

The baseline profile of H.264 standard is typically considered as the simplest profile in H.264/AVC. B-slices, weighted prediction, interlaced coding, CABAC, SP/SI slices and slice data partition are not included in the baseline profile. Low delay requirements system is the first target application with this profile. The implementation of our project uses baseline profile. CAVLC is not an essential unit in the research and we are not intending to discuss entropy coding algorithms. So just state-of-art of CAVLC is designed and only PPE part of CAVLC will be implemented in this version of H.264 encoder.

Table 5.1 Normal application requirements for H.264 profiles

<table>
<thead>
<tr>
<th>Application</th>
<th>Requirements</th>
<th>H.264 Profiles</th>
<th>MPEG-4 Profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast television</td>
<td>Coding efficiency, interlace, reliability (over a controlled distribution channel)</td>
<td>Main</td>
<td>ASP</td>
</tr>
<tr>
<td>Streaming video</td>
<td>Coding efficiency, reliability (over a uncontrolled packet-based network channel), scalability</td>
<td>Extended</td>
<td>ARTS or FGS</td>
</tr>
<tr>
<td>Video storage and playback</td>
<td>Coding efficiency, interlace, low-complexity encoder and decoder</td>
<td>Main</td>
<td>ASP</td>
</tr>
<tr>
<td>Videoconferencing</td>
<td>Coding efficiency, reliability, low latency, low-complexity encoder and decoder</td>
<td>Baseline</td>
<td>SP</td>
</tr>
<tr>
<td>Mobile video</td>
<td>Coding efficiency, reliability, low latency, low-complexity encoder and decoder, low power consumption</td>
<td>Baseline</td>
<td>SP</td>
</tr>
<tr>
<td>Studio distribution</td>
<td>Lossless or near-lossless, interlace, efficient coding</td>
<td>Main</td>
<td>Studio</td>
</tr>
<tr>
<td></td>
<td></td>
<td>High Profiles</td>
<td></td>
</tr>
</tbody>
</table>

26
5.2 System Architecture

The approach we used is to assign an SPE to an encoder to complete a unit of work and then to be reassigned by PPE. The SIMD model with 6 SPEs and 1 PPE is described in Figure 5.1. The H.264 encoding process is divided into four basic functions: motion estimation (ME), intra encoding (Intra), transform and quantization (PI), and CAVLC. Each of these functions operates on one partition of a frame at a time and constitutes the unit of work for an SPE except CAVLC. The SPE is passed a command block which contains the operation to be performed and pointers to the frame buffers on which it is to perform the function. As the SPE completes the function on a frame, it notifies the PPE which reassigns it to the next pending work unit. In this way, any mix of frame rates, frame sizes, and IPB stream structures can be managed automatically by the Cell.
As mentioned, the encoding process was divided into four functions. The first, motion estimation (ME) is the most difficult to implement in an SPE due to the large amount of buffering this function requires. In our project, the range for motion estimation will be studied and researched with a comparison with previous work. Our motion estimation search range is 64, four times bigger than the previous version. Therefore 20736 \((16 \times (4 \times 2 + 1) \times 16 \times (4 \times 2 + 1))\) comparisons will compute for one macroblock motion estimation if the full search algorithm is applied. The computing complexity of 64 search range is 9 times as big as that of 16 search range. The second function Intra-prediction will select one prediction mode and one prediction direction. The sum of Absolute Errors (SAE) is used to find the best combination mode/direction. The mean numbers of operations per macroblock for the H.264 full intra esti-
mation approach are: 33326 additions, 1075 multiplications, 14342 comparisons and 5217 divisions. So the total operation per MB in the Intra-prediction is 53960, which has the same complexity as in motion estimation. As with the ME function, transform and quantization (PI) process one or several rows of macroblocks after intra/inter prediction is done. The intra-prediction mode or motion compensated prediction vector is provided to PI together with the macroblock data. If the macroblock has completed PI function, then it can be used as a reference macroblock for the next frame. The last function is the entropy coding, which is used as CAVLC in our encoder. The motion vectors and coefficients will be encoded in this function. Normally it is not suited to a vector processor like SPEs to do this entropy coding. We can use PPE to process the CAVLC after all macroblock are processed. Even the PPE doesn't have enough power to handle the encoded video stream in real-time, we can prove that CELL is available to do entropy coding. Related research showed that the PPE could get the same amount of time in entropy coding as the SPE if the optimized program structure was applied [10].

5.3 Frame Partition and Scheduling

Remember that only 256KB Local Store (LS) can be used for both instructions and data in each SPE. We must put the working set of an SPE program resides completely in the 256 KB local store. Normal high definition video frames are more than 3Mbyte if it's uncompressed. There is no way to put one entire HD frame into the LS. We have to assign the usage of LS very precisely, and only scheduled data can be transferred into the respective SPE's LS from the external main memory. Programming techniques such as data overlay and code overlay can be used if necessary.

5.3.1 PPE Frame Reading

YUV420 is a planar format, in which 420 presents each pixel has its own Y value and a pair of U and V entry applies to one 2x2 pixel block. In other words, the 2x2 block shares one same U and V values. Since Y value is corresponding to each individual pixel, 4 bytes will be used to store Y values in one 2x2 block. Both U and V are 1byte, 2 bytes is needed to store U and V values in one 2x2 block.

Take one 6x4 frame as example in Figure 5.2.
Figure 5.2  One 6x4 frame in YUV420 format.

In YUV420 stream, this 6x4 frame will take 36 bytes. The data is presented as three parts.

- **Y Component: (0-23th bytes)**

```
<table>
<thead>
<tr>
<th>Y0</th>
<th>Y1</th>
<th>Y2</th>
<th>Y3</th>
<th>Y4</th>
<th>Y5</th>
<th>Y6</th>
<th>Y7</th>
<th>Y8</th>
<th>Y9</th>
<th>Y10</th>
<th>Y11</th>
</tr>
</thead>
</table>
```

- **U Component: (24th - 29th bytes)**

```
<table>
<thead>
<tr>
<th>U0</th>
<th>U1</th>
<th>U2</th>
<th>U3</th>
<th>U4</th>
<th>U5</th>
</tr>
</thead>
</table>
```

- **V Component: (30th - 35th bytes)**

```
<table>
<thead>
<tr>
<th>V0</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
</tr>
</thead>
</table>
```

In one video sequence, there is more than one frame stored in YUV420 format. The data stream is

\[
Y(\text{frame0}) \ U(\text{frame0}) \ V(\text{frame0}) \ Y(\text{frame1}) \ U(\text{frame1}) \ V(\text{frame1}) \ldots
\]

So when PPU read one frame from the original full HD video file, it uses 1920x1088 bytes, which includes the frame extending, to put all pixels' Y values in the frame, and then notices specified SPUs for processing. If all SPUs report that assigned macroblocks have been encoded completely, the PPE will skip 1044480 bytes U and V values and read the next frame Y values since the chroma is not encoded in this version of the encoder.
5.3.2 SPE Frame Partition

Consider that the high definition video resolution, 1080p (extended to 1088p), can be divided into 120 macroblocks wide and 68 macroblocks tall. Each macroblock has 16x16 byte data, 16 byte wide and 16 byte tall. There are 5 SPUs processing in a round-robin manner. If the DMA efficiency is added as the realistic requirement, one macroblock slab has to be divisible by 128 byte, which is the CELL DMA alignment.

5.3.2.1 I Encode Frame Partition

In I frame encoding we can get large LS size to process I frame macroblocks without the huge reference frame data. To reduce the non-alignment DMA, 68 horizontal slabs of 8160 macroblocks in one frame are set. Each slab contains one row of I frame macroblocks, 120 macroblocks in total. 17 slabs form one partition that will be encoded as one slice in H.264. Generally, there will be four slices in the encoded I frame. The overview of partitions is shown in Figure 5.3. Each color cell presents one macroblock slab, 120x1 macroblocks. The last number presents double buffer index in DMA.

![Figure 5.3 Horizontal encode frame partitions for SPUs.](image-url)
One SPE reads in a slab through DMA to main memory and copies upper encoded adjacent pixels from previous buffer (0/1), processes all macroblocks inside, and writes out the encoded macroblocks back to CELL main memory. Each SPE only reads the next slab in the partition assigned to it by PPE in the beginning. If double buffer is considered in I frame DMA, the buffer used in the local store is \((120x1x256 + 120x16)x2 = 64K \text{ bytes, which is a small size.}\)

### 5.3.2.2 P Encode Frame Partition

We divide 8160 macroblocks in one frame into 5 vertical partitions; each of them is processed by one SPE. The frame partition is 24 macroblocks wide. DMA alignment restricts the maximum number of partitions to be 15 (1920/128). 15 can be factored into two primes, 3 and 5. So we can use 3 SPEs to deal with 40 macroblocks per partition or 5 SPEs to process 24 macroblocks per partition. 40 macroblocks per partition needs large buffers in SPE that will extend the LS size. With the inheritance from the previous work, we use 24 macroblocks per partition and 2 macroblocks tall to form one slab. Thus there are 170 of these macroblock slabs in one 1080p frame. Each SPU will be assigned to work on two vertical adjacent slabs of macroblocks at a time for the DMA efficiency issue. It will hide the DMA time of reading the next macroblock slab from external main memory and the DMA time of writing the encoded macroblock back to external main memory while the new slab is being processed by the SPU. The Figure 5.4 shows the processing order of macroblock slabs using 5 SPEs.

![Figure 5.4 Vertical encode frame partitions for SPUs.](image-url)
In Figure 5.4 each cell presents one macroblock slab, 24x2 macroblocks.

### 5.3.2.3 P Reference Frame Partition

Above sessions describe the source frame partitions; here we will give the reference frame partition design. Each reference frame partition is based on the corresponding P encode partition. Comparing with 1920 byte Y values of reference pixels in one I partition, the P reference partition has a search range of 64x64 which need to be extended. It can be calculated as the sum of the same P encode frame slab (24x2x256 bytes) and enlarged 4 macroblocks in each direction ((24x4x2 + 4x10x2)x256 bytes), which equals 80K bytes. Double buffering requires two frame partitions in LS. So the minimum memory cost for one P macroblock slab is (24x2x256 + 80K) x2 = 184K bytes. We tested old version encoders, then got the result that the average executable code size on each SPE is less than 60K bytes. We estimated that our new encoder would not extend more than 10K bytes considering the complexity of Intra-encoding. The LS usage will barely fit into the maximum 256K bytes of LS.

### 5.4 Communication and DMA

PPU manages all communications between PPU and SPUs. It gives commands through mailboxes and channels. In this encoder version, it is the PPU's work to inform all SPUs which type of encoding is choosing, Intra or Inter. An enumerator structure is used to number all commands. Each time PPU sends an enumerator value to a mailbox with specified channel, and then waits for the replied value from the SPU associated with this channel. When a SPU encoder starts to run in the SPE, it steps into an infinite loop that waits for the command from PPU. Once the SPU reads values from the mailbox, it processes this value as the command given by PPU. All commands are shown in Table 5.2. Commands with the value less than 10 belong to SPE program environment operations and commands with the value more than 9 belong to frame encoding operations. More commands can be added easily in future.

<table>
<thead>
<tr>
<th>Command</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT_SPE</td>
<td>0</td>
<td>Clean memory and exit SPE</td>
</tr>
<tr>
<td>LOAD_CONTEXT</td>
<td>1</td>
<td>Wait for all data is ready</td>
</tr>
<tr>
<td>FLUSH_SPE</td>
<td>2</td>
<td>Read frame parameters</td>
</tr>
<tr>
<td>INTRA_ENCODE</td>
<td>10</td>
<td>Intra predict and encode</td>
</tr>
</tbody>
</table>
Inside the frame encoding operations, recursive related commands can be used to complete a combination of encoding work. A SPU will wait for the frame partition assignment command when it receives INTRA_ENCODE or MOTION_ESTIMATE command. The frame partition id is written to the mailboxes. Once a SPU read the partition id from the mailbox, it computes prediction information and encodes macroblocks in the partition. Besides that, inter prediction assigns macroblock slabs to fixed SPUs.

There is no way to optimize DMA requests for two P encode frame buffers. We shall focus on P reference frame reusing. In order to reuse the P frame reference frame partition, in the double buffering the common data between the first buffer (buf0) and the next buffer (buf1) is analyzed and stored in specified LS locations. Different parts of the reusing are described in Figure 5.5.

![Diagram](image)

Figure 5.5 Vertical reference frame partitions for SPUs.
In Figure 5.5, lighter gray cells present the private search macroblocks and darker gray cells present the common search macroblock for reusing.

This kind of reference frame reusing requires the macroblock slabs are all processed in SPU-order. Each SPU is assigned to one specified reference partition and only processes the corresponding encode slab. After one macroblock slab (slab B) is processed, the SPU resource manager in PPU does not assign another slab that isn't in the assigned partition. The next slab with index id B+5 will be loaded into local store and only the private macroblocks in the reference partition should DMA. The common macroblocks can be memory copied directly from slab B buffer when slab B buffer is ready to be processed. To synchronize the slab assignment, the PPU will hold the working SPU until the current task is completed, including prediction, transforming, common reference data coping, etc. A PPU-SPU_1 double buffering DMA data flow chart is shown in Figure 5.6.
Chapter 5  Design and Implementation

Send out command MOTION_ESTIMATION;

Assign partition 1 to SPU_1;

Assign slab M to SPU_1;

Any slab in partition?

Yes

Double buffers are ready.

No

No more slab to encode.

Read command;
Prepare for motion estimation;

Read slab 1;
Leave one slot in mailbox;

Read full reference frame for slab 1;
Leave one slot in mailbox;

Read part reference frame for slab 6;
Leave one slot in mailbox;

Read slab 6;
Leave one slot in mailbox;

Compute ME for all macroblocks in slab M -5 with buffer X;

Read slab M;
Leave one slot in mailbox;

Read part ref frame for slab M;
Switch buffer X;

Confirm DMA encode and reference buffer X is completed;

Compute ME for all macroblocks in slab M -5 with buffer X;

Copy the common ref frame;
Write back prediction results;

Read slab M;
Leave one slot in mailbox;

Read part ref frame for slab M;
Switch buffer X;

Confirm DMA encode and reference buffer 1 is completed;

Compute ME for all macroblocks in buffer 1;

Write back prediction results;
Chapter 5  Design and Implementation

Figure 5.6  Flow chart of PPU-SPU double buffering DMA example.

5.5 SIMD implementation of I-frame Encoding

In H.264 Intra frame encoding, it defines 4 different modes for intra_16x16 prediction and 9 different modes for intra_4x4 prediction. Every encoding block will calculate all candidate blocks to find the least residual signal for the best prediction mode. The general number of comparing computations is 4x16x16 + 9x4x4x16, around 13 SADs of Inter motion estimation. In many cases it needs more processing cycles than motion estimation in SIMD processors. Here we use full block estimation that checks every candidate block for each macroblock. We know there are various intra-encoding algorithms working well in ordinary general CPUs like x86. None of them is available for SIMD Intra encoding. One significant disadvantage is too much branches in the algorithm. So we choose the standard encoding procedures in [6] and left this big branch missing challenge to the coming version.

If a macroblock is to be encoded as an intra macroblock or in intra mode, prediction macroblocks are computed based on encoded macroblocks and prediction modes. Then the Sum of Absolute Errors (SAE) is calculated between the original macroblock and the generated intra prediction macroblock. All modes' SAE values are compared and the mode with the minimum SAE is the final intra encoding mode.

The basic workflow is shown below.

1. Read macroblock position information
2. Compute intra_16x16 DC mode prediction macroblock
3. Compute intra_16x16 vertical mode prediction macroblock
4. Compute intra_16x16 horizontal mode prediction macroblock
5. Compute intra_16x16 plane mode prediction macroblock
6. Choose one 4x4 macroblock from the encoding 16x16 macroblock
7. Compute intra_4x4 DC mode prediction macroblock if available
8. Compute intra_4x4 vertical mode prediction macroblock
9. Compute intra_4x4 horizontal mode prediction macroblock
10. Compute intra_4x4 diagonal down left mode prediction macroblock
11. Compute intra_4x4 diagonal down right mode prediction macroblock
12. Compute intra_4x4 vertical right mode prediction macroblock
13. Compute intra_4x4 horizontal down mode prediction macroblock
14. Compute intra_4x4 vertical left mode prediction macroblock
15. Compute intra_4x4 horizontal up mode prediction macroblock
16. Calculate SAE, choose the minimum value as intra_4x4 prediction SAE
17. Calculate sum of 16 SAEs of intra_4x4 macroblocks and compare with 4 intra_16x16 SAEs to find the minimum one.

18. Copy the prediction macroblock with minimum SAE.

SAE can be calculated by the following equation:

$$\text{SAE} = \sum_{i=0}^{m} \sum_{j=0}^{n} |X(i, j) - Y_k(i, j)|$$

where $X(i, j)$ presents the original value of pixel $(i, j)$ and $Y_k(i, j)$ presents the $k$ mode estimation macroblock value of pixel $(i, j)$. To SIMD SAE calculation, we can take the same method from old Inter-frame SIMD SAD calculation. More detailed information about SIMD SAD can be found in [4].

Now we design the SIMD predict macroblock computation. There are four main types of most used functions in the intra prediction. The SPU supports operations only on quadword (128 bits) alignment because of the register file. The neighboring values for one 16x16 macroblock just fits two unsigned char vectors and only one pixel in the upper left corner is lonely to form a vector itself. Assume that upper neighboring samples are stored in vector unsigned char $u[0]$ and $u[1]$, $u[1]$ has main values. Left neighboring sample is stored in vector unsigned char $u[2]$. 16x16 prediction macroblocks are save in a big buffer pre_mb[] that the buffer index is the mode value.

### 5.5.1 Vertical Prediction

In this mode, the prediction macroblock is generated by upper neighboring $u[1]$.

For each vector $i$ in pre_mb[0]

* $(\text{pre_mb} + i) = u[1]$;

### 5.5.2 Horizontal Prediction

The left neighboring vector $u[2]$ is used to compute this prediction macroblock. Predefine patterns are applied to extract the value from $u[2]$.

For each vectors $i$ in pre_mb[1]

* $(\text{pre_mb}[1] + i) = \text{spu_shuffle}(u[1], \text{temp}, \text{patten}[i])$;

### 5.5.3 DC Prediction

It sums all values in $u[1]$ and $u[2]$ to have the mean, and extend this mean value to every
value position in the DC prediction macroblock. We use spu_sumb to join two 16 bytes vector together into a unsigned short vector us_sum, and shift the us_sum 3 times to add all values. Once we get the sum of all values in u[1] and u[2], the vector shifts 5 bits right for the mean.

```c
us_sum = spu_sumb(u[1], u[2]);
us_buf = spu_slqwbyte(us_sum, 8);
us_sum = spu_add(us_sum, us_buf);
us_buf = spu_slqwbyte(us_sum, 4);
us_sum = spu_add(us_sum, us_buf);
us_buf = spu_slqwbyte(us_sum, 2);
us_sum = spu_add(us_sum, us_buf);
us_res = spu_slqw(us_sum, 11);
us_res = spu_shuffle(us_res, temp, patten);
```

For each vector i in pre_mb[2]
```c
*(pre_mb[2] + i) = us_res;
```

### 5.5.4 Plane Prediction

Even only intra_16x16 has plane mode, other prediction modes like intra_4x4 diagonal right have the same SIMD structure as Plane mode. Plane mode prediction macroblock is the hardest one to compute. The equation can be found in the standard. Figure 5.7 demonstrates the Plane computing.
The neighboring vectors are shuffled first to transform into signed short vectors. The main operation right now is subtracting that can't be positive value or zero always. So it has to cast into the signed short vector to perform further operations. When ia, ib and ic vectors finish computations, the coming step requires them to be signed short type vectors. Then they are conversed
back to the signed short and multiply predefined constant vectors to generate different \((i,j)\) values. Finally, all data are added in prospective. We check whether the final value is in the zone of \([0,255]\) or not.

Loop unrolling will be considered in the implementation. Otherwise the number of dependency stalls will increase to eliminate the advantage of long 128 bits register file in SPU. Different data positions according to the mode can archive good dual cycles and low dependency.

### 5.6 Motion Estimation

To meet the LS limitation and real time requirement, a better motion estimation algorithm will be introduced. Consider the main performance metric we interest is the relationship between frame quality and the motion search range. So the refined block-based hexagon search algorithm is employed.

The motion estimation algorithm we used in this version is based on simple hexagon and previous version encoder. It calculates the SAD with the same code as the old one. The adjustments are the search forward control logic and reference manager. More SIMD motion estimation algorithm information and SAD calculation can be found in [4]

#### 5.6.1 Reference Management

As described in session 5.4, in double buffering loop, when SPU finishes the encode work of buf0 and is ready to process buf1, it should use SIMD memcpy to copy the common reference part directly from buf0, not DMA from the external main memory. If we can find the right tradeoff between memcpy and DMA, then CELL processor is working in a small stall waiting time.

Normal DMA latency in CELL is 50 ns to transfer 256 bytes. If a 3.2GHz CELL is used, the total cycle of DMA one reference frame slab is 51200 (= \(50 \times (32 \times 10 \times 16 \times 16 / 256) \times 3.2\)) cycles. If the memcpy is used, the best case of all dual cycles is 2560 (= \(32 \times 10 \times 16 \times 16 / 16 / 2\)) cycles and the worst case of all single cycles is 5120 (= \(32 \times 10 \times 16 \times 16 / 16\)) cycles. The memcpy cycle is only one tenth of the DMA cycle. Now we use memcpy to copy the common reference data. The memcpy needs 2048—4096 cycles to copy, DMA needs 10240 cycles. The memory loading cycle can be decreased from 51200 to 10240 cycles.

Memcpy is implemented as the following SIMD steps.
For each 8 vector i in common reference buffer
\[
\text{dst}[i] = \star((\text{vector unsigned char }\star)(\text{src}[i]));
\]
\[
\text{dst}[i+1] = \star((\text{vector unsigned char }\star)(\text{src}[i+1]));
\]
\[
\text{dst}[i+2] = \star((\text{vector unsigned char }\star)(\text{src}[i+2]));
\]
\[
\text{dst}[i+3] = \star((\text{vector unsigned char }\star)(\text{src}[i+3]));
\]
\[
\text{dst}[i+4] = \star((\text{vector unsigned char }\star)(\text{src}[i+4]));
\]
\[
\text{dst}[i+5] = \star((\text{vector unsigned char }\star)(\text{src}[i+5]));
\]
\[
\text{dst}[i+6] = \star((\text{vector unsigned char }\star)(\text{src}[i+6]));
\]
\[
\text{dst}[i+7] = \star((\text{vector unsigned char }\star)(\text{src}[i+7]));
\]

5.6.2 Motion Estimation Algorithm

The search pattern has an important influence on speed and quality performance in block motion estimation. Consider the original simple hexagon search algorithm, 32 (\(=4+16/2\times3+4\)) SADs are calculated for the worst search case with 16x16 search range. If the search range increases to 64x64, 104 (\(=4+64/2\times3+4\)) search points will be evaluated in 66 steps. We have to decrease the number of SADs to meet the cycle of DMA latency.

Here we propose a refined hexagon search algorithm. The forward distance of the refined one increases from 2 to 8. If no search point is better than the center one, the forward distance will be set as 2 for another ordinary hexagon searching. If the center is the best again, DS algorithm will be applied to check surrounding 4 points. With this algorithm, only 47 (\(=4+8\times3+6+3\times3+4\)) search points will be examined in the worst case. The proposed algorithm can be summarized in the following detailed steps.

Step 1) Seven hexagon search points with the distance of 8 are checked. If the center of the hexagon has the minimum cost, proceed to Step 3); otherwise, proceed to Step 2).

Step 2) Move the center to the minimum cost point in Step 1), check three hexagon search points with the distance of 8, identify the minimum cost point. If the minimum cost point is found to be at the center of the hexagon, go to Step 3); otherwise, repeat this step continuously.

Step 3) Move the center to the minimum cost point in Step 1) or Step 2), check six hexagon search points with the distance of 2, identify the minimum cost point. If the minimum cost point is found to be at the center of the hexagon, go to Step 4); otherwise, repeat this step continuously.

Step 4) The hexagon center moves to the minimum cost point in Step 3). Four DS check points are evaluated. The final minimum cost point will be generated among these five points in the small diamond and motion vector is calculated from original block to final point.
5.7 CAVLC

Since the heavy workload of 1080p CAVLC and CELL architecture, it is not suitable to implement paralleled entropy coding like CAVLC. We decide to implement CAVLC in PPE side using scalar instructions in non-real-time. This part is not our main concern in the encoder. We implement it for the purpose of the baseline profile.

There six parameters of CAVLC will be calculated as described in Table 5.3

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coeff_token</td>
<td>TotalCoeffs and TrailingOnes</td>
</tr>
<tr>
<td>trailing.ones</td>
<td>Sign of TrailingOne value</td>
</tr>
<tr>
<td>level_prefix</td>
<td>First part of code for nonzero coefficient</td>
</tr>
<tr>
<td>level_suffix</td>
<td>Second part of code for nonzero coefficient</td>
</tr>
<tr>
<td>total_zeros</td>
<td>The number of zeros after the first nonzero coeff</td>
</tr>
<tr>
<td>run_before</td>
<td>The number of zeros preceding each nonzero coeff</td>
</tr>
</tbody>
</table>

The CAVLC can be implemented in the following detailed steps.

Step 1) Encode both the total number of TotalCoeffs and TrailingOnes based on four look-up tables.
Step 2) Encode all TrailingOnes with a single bit in reverse order. 0 presents +, 1 presents -.
Step 3) Encode sign and magnitude of all remaining nonzero coefficient in the array in reverse order.
Step 4) Encode the num of all zeros before the last nonzero coefficient.
Step 5) Encode the number of zeros preceding each nonzero coefficient in reverse order, starting with the highest frequency.
This chapter describes the evaluation of the H264 project in the CELL. This chapter is organized as follows: First of all, the generic simulation environment is outlined. Then we introduce the test video sequences used in the benchmark. The third part discusses the actual evaluation.

6.1 Benchmark Configures

The benchmarking is performed in the IBM CELL Software Development Kit version 3.1 (SDK 3.1) and IBM Full-System Simulator version 3.0 for the Cell Broadband Engine Processor. Simple mode and cycle mode are applied to measure different system performances. In simple mode, the fastest mode, we test 1 second video sequence (30 frames) to measure overall performances. The first frame (frame 0) is encoded as I-frame always and all the rest frames are encoded as P-frame. When 30 frames are encoded completely, we use the configured JM16.2 and gnuplot to decode frames for PSNR and motion vector statistical performances. In cycle mode, we encode 1 I-frame (frame 0) and 1 P-frame (frame 1) to measure the key functions.

Three wide using benchmarking 1080p video sequences are selected as our main input data.
They cover most common life scenarios and different angles/motions. All of them are YUV420 format video recorded by Sony HDW-F900.

- Blue sky: The camera is on the ground to shot the sky upwards, rotating clockwise in the first 1 second. Top of two trees are in two corners respectively. The tree contains many detailed information due to lots of moving leaves. At the same time, there are only small differences in the sky. The video has a high contrast, tree are much darker than the sky in brightness.

- Sunflower: It presents a bee beating wings at the surface of a sunflower. The bee moves while the sunflower is shaking. Vivid sunflower, living bee and fast beating wing make the video full of different color and motions.

- Riverbed: It records a small stream. Two main objects are water and riverbed seen through the water. When the water moves, it changes its shape and causes brightness shifting of the reflected sunlight. It is very hard to encode.

Quantization parameter QP is 28 in all encoding frames.

### 6.2 Frame Performance

Here we give the frame related results using different performance metrics.

PSNR Peak Signal to Noise Ratio (PSNR) is widely used as a reconstructed picture quality measure. The signal is the the original YUV frame data, and the noise or difference is the error when we compress each frame. It can be expressed using the equation:

$$\text{PSNR} = 10 \log_{10} \left( \frac{2^n - 1)^2}{\text{MSE}} \right)$$

where $n$ is the number of bits used for each pixel, 8 in our case, and MSE (Mean Square Error) is the root mean square difference between the original and the reconstructed frame. The PSNR result is measured in dB. The human eye can notice a difference in PSNR of 0.5 dB. Matlab is used to calculate frame PSNR. Figure 6.1 shows the PSNR result of three video sequences in our benchmark.
Table 6.1 shows the bit size used by I frame and the average size of P frame. Both CA VLC enable and disable conditions are tested. Normally one frame is divided into three main parts of bit usage, mode selection bits, motion information and encoded macroblocks. P frame uses motion information to reduce the encoded macroblock size, so one P frame size is much smaller than one I frame. Both I frame and P frame are extremely large in Riverbed, and P frame is even bigger than its I-frame. Let’s look at Table 6.2. The picture contains vast detailed information so that most I frame macroblocks choose intra_4x4 for a better SAE. It’s the same in P frame when macroblock division is made. More sub-blocks need more motion vectors to remember or to decode. In riverbed, this kind of motion information becomes one of the major bit usage sources. This is what we called as “hard to encode”.

Figure 6.1  PSNR per frame in three videos
Figure 6.2 I frame of video sequence Sunflower
Figure 6.2(a) is the decoded I frame done by JM H.264 reference software. Since we didn’t save enough H.264 SPS/PPS parameter information, many default values must be configured for decoding process. We can zoom into the bee head part and shadow part in the flower, see Figure 6.2(b) (c). Macroblock with less change or smoothly-varying luminance are selected and encoded as Intra_16x16. The macroblock that contains large information can be divided into smaller Intra_4x4 blocks for a better matching.

Table 6.1  The average bit size for one encoded frame. Metric: K bits

<table>
<thead>
<tr>
<th></th>
<th>Bluesky</th>
<th>Sunflower</th>
<th>Riverbed</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-frame with CAVLC</td>
<td>861.08</td>
<td>607.51</td>
<td>1355.52</td>
</tr>
<tr>
<td>I-frame without CAVLC</td>
<td>896.38</td>
<td>631.52</td>
<td>1404.45</td>
</tr>
<tr>
<td>P-frame with CAVLC</td>
<td>144.17</td>
<td>169.43</td>
<td>1858.13</td>
</tr>
<tr>
<td>P-frame without CAVLC</td>
<td>148.35</td>
<td>174.64</td>
<td>1941.09</td>
</tr>
</tbody>
</table>

Table 6.2  Two types Intra macroblocks for three videos. Metric: macroblocks(16x16)

<table>
<thead>
<tr>
<th></th>
<th>Bluesky</th>
<th>Sunflower</th>
<th>Riverbed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intra_4x4</td>
<td>3249</td>
<td>3470</td>
<td>7435</td>
</tr>
<tr>
<td>Intra_16x16</td>
<td>4911</td>
<td>4690</td>
<td>725</td>
</tr>
</tbody>
</table>

Table 6.3  The average p-frame PSNR in three videos. Metric: dB

<table>
<thead>
<tr>
<th></th>
<th>Bluesky</th>
<th>Sunflower</th>
<th>Riverbed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Version</td>
<td>38.09</td>
<td>40.73</td>
<td>35.57</td>
</tr>
<tr>
<td>Our New Version</td>
<td>38.16</td>
<td>41.04</td>
<td>36.11</td>
</tr>
<tr>
<td>Improvement</td>
<td>0.34%</td>
<td>0.76%</td>
<td>1.52%</td>
</tr>
</tbody>
</table>

We write our own program in Linux/C to extract the motion vector information from an encoded P-frame. The old version of the encoder has a motion search range of [-15, 15], which is shown in Figure 6.3(a), and our new version encoder increases the motion search range up to [-63, 63] captured in Figure 6.3(b). Both (a) and (b) is the bee head part of frame 2 in sunflower sequence. The background – sunflower – has high similar partitions and motion vectors between two examples. Most differences come from the bee, especially the wing. The wing beats quickly in a high frequency for a working bee. So the bee wing pixel distance between frame 1 and frame 2 is much larger than other parts of the bee. In such situation, large motion search
range is needed. Table 6.3 shows the average PSNR for all P-frames in all three video sequences. P-frame only gets 0.34% --1.52% improvement even the motion search range is 16 times bigger. We think the main reason of the improvement is 64x64 motion search range indeed finds out a better matching block with less SAD for fast moving blocks. Consider the ratio between the bee wing and whole frame, even we get some big improvements in the wing, the entire improvement is only less than 1%.

Additionally, a 64x64 search range requires high CPU workload and almost twice computing time as that of 16x16. Because of the same reason, new encoded P frame size is tiny smaller than old version one. Our new encoder can handles high speed movement video sequences very well. To low or normal speed, the improvement is not significant. We can say that 16x16 search range is enough for most Full High Definition videos.

![Figure 6.3 The motion vectors around bee body in Sunflower Video.](image)

**6.3 Cycle Performance**

We insert the simulator performance monitor into the code, and then execute the encoder in cycle mode. First we encode one complete I-frame (P0) in Sunflower. Key routine performances are measured by different cycle types and Cycle per Instruction (CPI) in Table 6.4.
Then we encode one P-frame (P1) from Sunflower. 170 macroblock slabs for one frame, divided into 5 partitions, each one assigned to a SPE. It means that each SPE processes 34 slabs of macroblocks. The Table 6.5 shows the total cycle information to encode one partition (34 macroblock slabs) in Mcycle.

Table 6.5  Cycle usage for one macroblock partition (1632 macroblocks per partition)

<table>
<thead>
<tr>
<th></th>
<th>Single Cycle</th>
<th>Dual Cycle</th>
<th>Nop Cycle</th>
<th>Dependency Stall</th>
<th>Total Cycle</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>one partition</td>
<td>12.95</td>
<td>5.18</td>
<td>1.41</td>
<td>1.1</td>
<td>20.72</td>
<td>1.16</td>
</tr>
</tbody>
</table>

Finally we get the 20.72M cycles for a SPE to complete to encode all P frame partitions assigned by PPE. If the CELL processor has a clock cycle of 3.2GHz, the result has shown that
our encoder can encode 30 frames in less than one second. In other words, our encoder encodes most 1080p Full HD video in real time. Consider that only 5 SPEs are used in this version, there is huge potential computing power for the CELL processor to do real time encoding in a faster and quicker way.

6.4 DMA Issues

After several tests, we have the conclusion that the DMA latency is not simulated in the IBM Full-System Simulator version 3.0. In a real CELL processor, different SPEs have different DMA latencies due to different distances from the memory controller. In other words, the physical position will affect the DMA latency. This kind of feature is not implemented in the simulator, so we cannot benchmark the DMA related performances. The benchmark can be done in PlayStation 3 in future.
In this thesis, we studied H.264 video prediction, especially intra prediction. 4 intra_16x16 prediction modes and 9 intra_4x4 prediction modes supply many combinations of different blocks of different sizes to find out the best matching block.

### 7.1 Conclusion

In this thesis we implement a H.264 baseline high definition encoder on CELL processor. We start from our old version of H.264/AVC video encoder, and design the new encoder architecture with data partition. Intra prediction, refined inter prediction and CAVLC are implemented. The benchmark confirms that the Cell BE processor has the ability to encode 1080p Full HD videos in real time using H.264 baseline profile.

By performing architecture optimization and algorithm optimization, our encoder archives a good encoding performance. Under the same complexity level, the intra prediction has significant frame quality improvement and higher CELL processor utilities than the inter prediction. At the same, we give the experiment results of different motion estimation search ranges and point out that 16x16 search range is enough for most videos.
7.2 Future work

A main challenge for the next version encoder is to reduce the branch miss and improve CPI in both intra and inter predictions. Due to the time limitation, we don’t refine all modules and intra prediction algorithm well enough. Better instruction optimization always leads to more computational abilities of the CELL processor.

The next interesting issue is the DMA performance in physical CELL processor. Since the new processor architecture is applied in the EIB bus, we would like to evaluate the real DMA performance and adjust our DMA/Processing ratio to get the maximum utility.
REFERENCES


