

Efficient H.264 Architecture Using Modular Bandwidth Estimation

Ruei-Xi Chen[†], Wei Zhao[§], Qinyi Liu[‡], Jeffrey Fan[§]

[†]Dept. of Computer Science and Information Engineering, St. John's University, Taipei, Taiwan

[§]Dept. of ECE, Florida International University, Miami, Florida 33174, USA

[‡]Dept. of Software Engineering, Beihang University, Beijing, 100081, China

ABSTRACT

Bandwidth is always one of the bottlenecks in System-on-a-Chip (SoC) systems. In this paper, we propose an efficient architectural design in analyzing the bandwidth of each component in an H.264 design. We decompose the entire H.264 system bandwidths into several modules with predictable coefficients. The derived equations may help designers understand the *real* cost of each hardware component, thus improving the efficiency of overall system. The main idea of this paper is to generate an H.264 architecture with all the desired features possible. If the model is unable to fit well in the overall system or subsystem, the designer can detect and modify the architecture in the early stage of the product development cycle, thus reducing the potential risk of system re-design.

General Terms

Algorithms, Performance, Design

Keywords

H.264 Architecture, SoC Design, Logical Design

1. INTRODUCTION

The emerging video coding standard H.264/AVC [1] which is developed by the ITU-T (International Telecommunication Union) and MPEG (ISO/IEC Moving Picture Experts Group), also known as MPEG-4 Part 10, provides a much efficient way in compressing the video as about 50% data size as it used to be in older standard. The video data has to be compressed because of the limited storage and the restricted transmitting bandwidth [2].

The similar situation also happens in chip design. The standard becomes more and more complicated as the compression ratio gets higher and higher. The sophisticated standard causes the high complexity of the hardware design. The entire system design (i.e. SoC design) is much more unpredictable than ever. According to the instruction profiling with HDTV1024P (2048 × 1024, 30fps) specification, H.264/AVC decoding process requires 83 Giga-Instructions Per Second (GIPS) computation and 70 Giga-Bytes Per Second (GBPS) memory access. In H.264/AVC encoder, up to 3600 GIPS and 5570 GBPS are required for HDTV720P (1280 × 720, 30fps) specification [3].

Apparently, although many excellent works on H.264 integer motion estimation schemes have been proposed [4, 5, 6], there still exists a wide gap for hardware to match up with

the efficiency. The software solution is always the better solution (in consideration of the cost), when ideally there is no computing time limitation. However, realistically there exist applications with time constraint, or even worse, need *Real Time* processing. In this condition, we could only use the hardware solution. The H.264 hardware-software co-existing

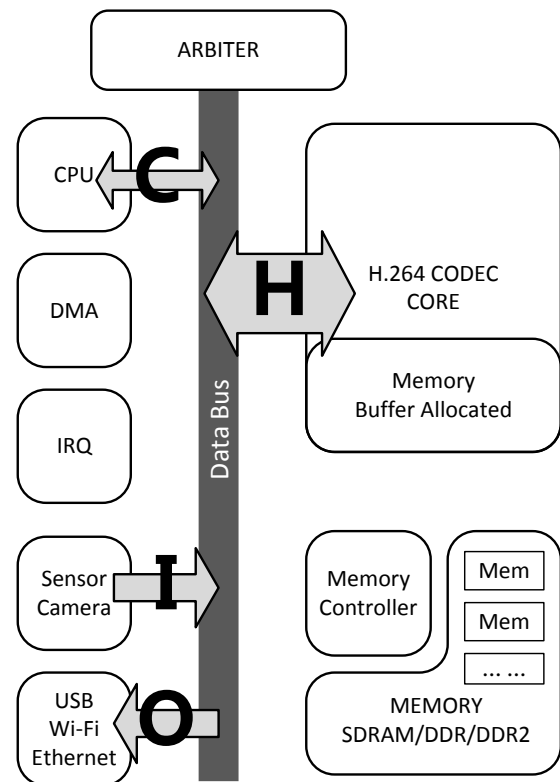


Figure 1: H.264 Based SoC Architecture Sketch

solution is a hot topic in recent research. Undoubtedly, however, the H.264 implementation resides in the System-on-Chip (SoC). As the High Definition (HD) standard of video is more and more popular around us today, the H.264 CODEC based SoC hardware solution is also dramatically increasing. In this paper, we will discuss a simple H.264 CODEC based SoC chip model, where the bandwidths are calculated in each component individually. We will show that the results base on the proposed approach improve the efficiency in overall system design cycle.

2. H.264 BASED SOC ARCHITECTURE

Figure-1 is a simple demonstration of H.264 based SoC architecture. It contains some CPU (such as ARM9), a BUS system (i.e. AMBA bus), and some accessories allocated to the system. Different architectural designs involve many aspects, but the overall architecture is almost identical. This basic illustration reveals some fundamental understanding of an overall SoC bandwidth construction (that is, some input and output of the modules):

1. C stands for the CPU software cost for the application. Usually, companies should have an estimate of their C s in the specific application such as mp3 decoders. Thus, we will not list them in our proposed prediction table.
2. I and O stand for the Input (sensor) and Output (such as USB controller, ethernet).
3. H is the H.264 I/O bandwidth. It is the key component of the proposed bandwidth analysis. Different algorithms may have different bandwidth requirements. The designer may need to consume more built-in logic or memory resources in order to satisfy the bandwidth requirements of the applications.

Table 1 lists a few examples of the constraints imposed for the frequency of special characters in SoC system: These constrains will need to be considered in the proposed architectural design.

Table 1: Frequency of Special Characters

| Net Meeting | Mobile Multimedia | Home Video |
|----------------|-------------------|------------------|
| Mobile Meeting | MP4 Player | HD Digital Video |
| Vision Call | Mobile Video | HD Television |
| CIF/QVGA 30fps | QVGA/VGA 30fps | 720p/1080p 60fps |

2.1 Bandwidth Model of SoC

Based on H.264 system application as shown in Figure-1, we can assume the system bus bandwidth cost function as:

$$B_{sys} = C + I + O + H \quad (1)$$

C exists in every system since it is the control line from the CPU. As we have mentioned, every company has different C models based on different applications. We cannot easily estimate universal C value, so we just put an C here till it is really used in some specific application. The I , H , and O are all connected to the memory side. Hence, the bandwidth of the memory should be $B_{mem} = I + O + H$.

Specifically, the I is the input source of the system. In *Real Time* specific H.264 application, it stands for the input source of the video data from the sensor or camera. The O component is the output compressed data of the encoder. Generally speaking, O is the result of I compressed with H.264 algorithm and O is much smaller than I , usually about 1/50 or 1/100. As the bandwidth goes really high in the H.264 CODEC, maybe several times, or even several tens of times of I , O can be neglected. Hence, H is the dominant term which affects the bandwidth of SoC. Let's describe the H part in the next section.

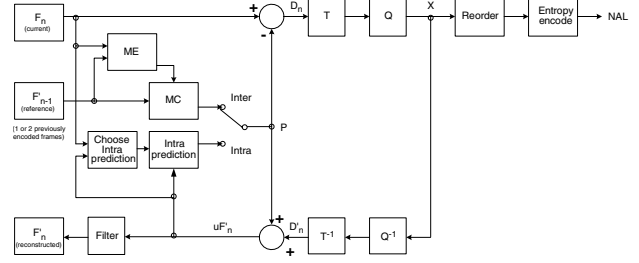


Figure 2: H.264 Encoder Architecture

3. BANDWIDTH MODEL IN H.264

Since H is the total bandwidth of the H.264 module, as shown in the Figure-2, it can be split into several parts in order to measure the whole bandwidth of H . We could see that our input is the sum of the one from the *Current Frame*, from inter-estimate unit (Motion Estimation) and intra-estimate unit. The output includes two parts: the Network Abstraction Layer (NAL) output derived from H.264 compression and the *Reference Frame* output for the compression afterwards. That is:

$$H = H_{in} + H_{inter} + H_{intra} + H_{out} + H_{ref} \quad (2)$$

The H_{in} and H_{out} of the equation is the general input and

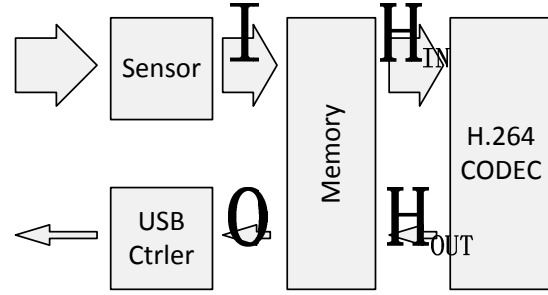


Figure 3: System Input and Output

output of the H.264 algorithm. As shown in the Figure-3, the I and O may not be exactly the same as H_{in} and H_{out} because of the buffer effects of memory. Generally speaking, however, the costs of bandwidth in H_{in} , H_{out} , I , and O are the same. The H_{ref} part is the reference building part. It is the part that H.264 CODEC sends the reference frame to the memory. The reference frame's generation is about the same as the current frame inside our module except in the opposite direction. Hence, the H_{ref} is the same as H_{in} .

3.1 The H_{inter} Bandwidth Estimation

In the H.264 ME algorithm, we process every frame not by pixels, but Macro Blocks (MB). The MB is a block contains 16x16 pixels. Furthermore, with technique of the partition of a whole frame in MB, we could find the MB which is most likely to be the current MB in the reference frames. However, it is too hard to search an exact MB in the whole frame itself. Usually, we search in a *Search Window* which contains many candidate MBs in the reference frames. Such search methods fall into two categories: the full search block

matching algorithm (FSBMA) and the variable block matching algorithm (VSBMA). FSBMA is the algorithm which searches for the exactly same 16x16 pixel block for the current MB in the full **Search Window**. Almost all the hardware solutions are based on the FSBMA because of its good quality and regular computation. For FSBMA, there are several searching methods. [7],[8],[9],[10],[11],[12] These architectural works are significant because all of them shed some light on the weights of hardware cost in order to save the analysis time in one way or the other. However, in terms of the bandwidth, when using full-search algorithm, they intend to have the same bandwidth outside the ME part. We will now start our bandwidth analysis. Here is a single frame model which consists of $N \times M$ Macro Blocks:

$$\begin{pmatrix} MB_{(0,0)} & MB_{(1,0)} & MB_{(2,0)} & \dots & \dots & MB_{(M,0)} \\ MB_{(0,1)} & MB_{(1,1)} & MB_{(2,1)} & \dots & \dots & MB_{(M,1)} \\ MB_{(0,2)} & MB_{(1,2)} & MB_{(2,2)} & \dots & \dots & MB_{(M,2)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ MB_{(0,N)} & MB_{(1,N)} & MB_{(2,N)} & \dots & \dots & MB_{(M,N)} \end{pmatrix} \quad (3)$$

All the MBs in current frames are going to be the primary base which will be taken and searched in the **Search Window** of one or multiple **Reference Frames**. We calculate the Sum of Absolute Differences (SAD) in each of the result and choose for the best difference which contains the least information and send them to the next procedure.

For the reference frames, assume that the **Search Window** is an $m \times n$, ($m \leq M, n \leq N$) MB window. Then, we will need to do the full 16×16 pixels search in that window:

$$\begin{pmatrix} \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \begin{pmatrix} MB_{(i,j)} & \dots & MB_{(i+m-1,j)} \\ \dots & \dots & \dots \\ MB_{(i,j+n-1)} & \dots & MB_{(i+m-1,j+n-1)} \end{pmatrix} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \quad (4)$$

The SAD is the difference between the current MB and the 16×16 pixels Block which is selected from the reference frame(s).

$$SAD(x, y) = \sum_{a=0}^{15} \sum_{b=0}^{15} Distortion(a, b, p, q) \quad (5)$$

p and q are the pixel distance from current pixel to the reference pixels, and ($0 < p \leq 15, 0 < q \leq 15$).

$$Distortion(a, b) = |cur(a, b) - ref(a + p, b + q)| \quad (6)$$

Use this SAD equation, we could get all the SADs from each point inside the **Search Window** we have selected. However, when the current MB shifts, the **Search Window** shifts, too. The current MB shifts from $MB_{(a,b)}$ to $MB_{(a+1,b)}$ and our **Search Window** is going to shift horizontally for one MB.

The original summation of *memorized* MBs in the ME module is:

$$\sum (orig) = \sum_{x=i}^{i+m-1} \sum_{y=j}^{j+n-1} MB_{(x,y)} \quad (7)$$

And, the new summation of *memorized* MBs in the ME module is:

$$\sum (curr) = \sum_{x=i+1}^{i+m} \sum_{y=j}^{j+n-1} MB_{(x,y)} \quad (8)$$

Comparing the original Search Window and the Current Search Window which is shifted, we have:

$$\sum (curr) = \sum (orig) - \sum_{y=j}^{j+n-1} MB_{(i,y)} + \sum_{y=j}^{j+n-1} MB_{(i+m,y)} \quad (9)$$

$$\Delta = \sum_{y=j}^{j+n-1} MB_{(i,y)} - \sum_{y=j}^{j+n-1} MB_{(i+m,y)} \quad (10)$$

We can see that the $\sum (curr)$ and $\sum (orig)$ are the memory we used inside the MB process. We have thrown in a group of MBs by adding the new group according to the shift of the current MB. Hence, the Δ is the Bandwidth we are looking for. The Δ consists of two parts. The first one is obviously the memory we are going to dump, thus do not need to count them in. The second part is the n MBs which will take a new column on the right side of our Search Window. So, the bandwidth cost when we shift the current MB into another, is n MBs. In the situation of r Reference Frames to be considered, the Δ becomes $n \times r$ MBs.

Since the input bandwidth of the H.264 CODEC is 1 MB at a time, we simply define that the bandwidth of the input stream as $1X_{spd}$. This could help us understand the concept of bandwidth increase. Here is the example:

Consider a "Chroma Format: 4:2:2, Frame Rate: 30f/s and Resolution is VGA (640 by 480)" video, its $1X_{spd}$ is:

$$CF \times FR \times R = 2 \times 30 \times 640 \times 480 = 18.432\text{MB/s} \quad (11)$$

For a "Chroma Format: 4:2:0, Frame Rate: 60f/s and Resolution is VGA (1920 by 1080)" video, its $1X_{spd}$ is:

$$CF \times FR \times R = 1.5 \times 60 \times 1920 \times 1080 = 186.624\text{MB/s} \quad (12)$$

Thus, the calculated H_{inter} becomes $(n \times r)X_{spd}$. With a unit of $1X_{spd}$, it can fit all the conditions in different resolutions or frame rates, even with different chroma formats. Different algorithms seldom matter the bandwidth of ME block much because the point they give out the algorithm is to increase the logical efficiency (reduce the logical elements) or to decrease some memory block for the hardware designs. In consideration of the bandwidth of the H.264 CODEC, especially the ME part, it is also the way for us to reduce the bandwidth at the cost of losing some cache memory.

Our SRAM memory is the $\sum (curr)$ and $\sum (orig)$ as we discussed previously, i.e. $(m \times n)$ MB. If we increase them into $(n - 1)$ lines of MBs, what will happen?

Figure-4 is the method we used in the last few paragraphs. It contains only $(m \times n)$ MBs as we saw in the diagram. The

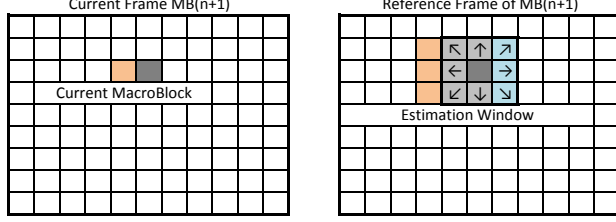


Figure 4: Cache Min Algorithm

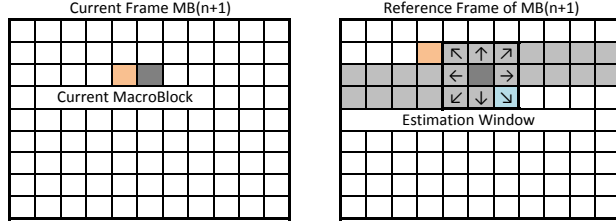


Figure 5: Bandwidth Min Algorithm

Figure-5 is the improved method in the bandwidth consideration. However, we will need to implement with more memories (caches) on the chip. The original summation of memorized MBs for the new Bandwidth Min algorithm becomes:

$$\sum(orig) = \sum_{x=i}^{M-1} MB_{(x,j)} + \sum_{x=0}^{M-1} \sum_{y=j+1}^{j+n-2} MB_{(x,y)} + \sum_{x=0}^{i+m-1} MB_{(x,j+n-1)} \quad (13)$$

Similarly, the new summation of memorized (cached) MBs for the new Bandwidth Min algorithm is:

$$\sum(curr) = \sum_{x=i+1}^{M-1} MB_{(x,j)} + \sum_{x=0}^{M-1} \sum_{y=j+1}^{j+n-2} MB_{(x,y)} + \sum_{x=0}^{i+m} MB_{(x,j+n-1)} \quad (14)$$

Compare the original Search Window and the Current Search Window which has been shifted:

$$\Delta = MB_{(i,j)} - MB_{(i+m,j+n-1)} \quad (15)$$

Since the Δ is only 1 Macro-Block, with r Reference Frames ME subsystem, the cost function of bandwidth is extremely low and it is just rX_{spd} . Put it into the 1080p example, if the Search Window is 8 by 5 MB, and the $1X_{spd}$ is equal to 186.624 MB/s, as we have discussed in the eq. (12), with 2 reference frames used. The total benefit in the bandwidth side is:

$$(5 \times 2 - 2)X_{spd} = 1492.992\text{MB/s} \quad (16)$$

However, on the other side, the cache cost becomes:

$$\begin{aligned} (8 \times 5)MB - [(\frac{1920}{16} - 8) \times (5 - 1) + (8 \times 5)]MB \\ = 448 \times (16 \times 16)_{\text{pixel}} \times 1.5_{\text{Byte/pixel}} \\ = 172,032\text{Byte} \end{aligned} \quad (17)$$

One may notice that 1492.992 MB/s bandwidth is traded for 172.032 KByte of SRAM. This sounds like a good deal! However, different applications may have different targets to meet. Nevertheless, the designer may consider and analyze the overall H.264 implementation to determine whether it is a good plan to start with.

3.2 The H_{intra} Bandwidth Estimation

The *Intra Prediction* [2] for an H.264 does not need that much of data for calculation. The hardest part for intra prediction is to compute in parallel manner. The only thing we need to cache is the bottom line of 2 upper MBs and the right line of the left MB of the current MB. As we described in the inter-prediction section, we have given two basic algorithms to describe the proposed evaluation theory. They are shown in the Figure-6 and the red spot line is the place we have to cache or read-in. Using the same model as we mentioned previously in eq. (3) and eq.(4), we could easily re-calculate our memorized memory and the Δ memory which is the bandwidth.

For the Cache-Min algorithm (Figure-6 Left):

$$\Delta = 16_{\text{pixel}} \times \frac{1}{256 \text{ MB/pixel}} = \frac{1}{16} \text{ MacroBlock} \quad (18)$$

Under the r Reference Frames situation, it only needs $\frac{r}{16}$ Macro-Blocks bandwidth. Even there are 16 reference frames, we only need $1X_{spd}$ of Bandwidth. If it is not counted for the cache size of Current Macro Block itself (since it was counted in the inter mode), the current need for the cache size becomes $3 \times 16 = 48$ pixels.

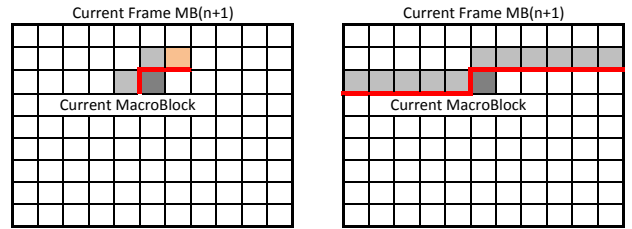


Figure 6: 2 Algorithms for the Intra Prediction

For the Bandwidth-Min algorithm (Figure-6 Right):

We do not need any extra bandwidth in the Bandwidth-Min algorithm in intra prediction. The current Macro-Block will be the left MB in next MB's procedure, and all the candidate pixel we wanted has already been in our cache. Our cache size is not so huge too, it is about 1 line of pixel storage: $1920 + 16 = 1936$ pixels (assuming that it is a 1080p resolution video), with the SRAM memory size of less than 4KB.

Table 2: Different Application Models for Analysis

| Resolution | FR | Color-Format | 1Xspd |
|------------------|----|--------------|-------------|
| 320×240(QVGA) | 30 | 4:2:2 | 4.608MB/s |
| 640×480(VGA) | 30 | 4:2:2 | 18.432MB/s |
| 720×480(NTSC) | 30 | 4:2:2 | 20.736MB/s |
| 1280×720(720p) | 30 | 4:2:2 | 55.296MB/s |
| 1920×1080(1080p) | 30 | 4:2:2 | 124.416MB/s |

4. EXAMPLES

As shown in Table-2, appropriate resolutions may be applied to different applications. Basically, the QVGA and VGA resolutions are for the handset electronic instruments. They may suffer from low qualities and low performance. Thus, it requires high ratio compressed code, faster transfer and lower frequency (i.e. handset batteries are limited). On the other hand, NTSC/PAL is popular for the SD/HD TV video signal resolutions. The broadcast signal can be reached on the modern cellular phones digitally. Furthermore, 720p/1080p are used in the full High-Definition (HD) TV resolutions, and they would be widely used as today's Standard-Definition (SD) television standard.

In the previous section, we have divided the algorithms into two corresponding categories, i.e. Cache-Min and Bandwidth-Min. However, they are not the only two algorithms in the H.264 implementation. As matter of fact, designers may not use only Cache-Min or Bandwidth-Min. Instead, they may use them combinatorially. No matter what algorithms they choose to use, the analysis of the bandwidth and cache cost in the model we've proposed may still apply to their designs. Such design examples are described as follows:

4.1 Cache-Min Algorithm Analysis

The bandwidth is directly proportional to the height of Search Window and the number of reference frames. The Cost of cache size is only related to the area size of Search Window. Consider C at 60MB/s as a standard simple cell-phone's CPU, the bandwidth can easily be calculated as:

$$\begin{aligned}
 B_{sys} &= C + I + O + H \\
 &= C + I + O + H_{in} + H_{inter} + H_{intra} + H_{out} + H_{ref} \\
 &= C + 1X + 0 + 1X + (r \times n)X + \frac{r}{16}X + 0 + 1X \\
 &= C + (3 + \frac{(16n+1)r}{16})X_{spd} \\
 C_{cost} &= C_{inter} + C_{intra} \\
 &= (m \times n)MBs + 48Pixels \\
 &= (256mn + 48)Pixels
 \end{aligned} \tag{19}$$

4.2 Bandwidth-Min Algorithm Analysis

Obviously, the bandwidth is only correlated to Reference Frame Number in this Bandwidth-Min algorithm. Thus, the cost of cache size is nearly the direct proportion to the

Bandwidth & Cache Cost in "Cache-Min" Approach

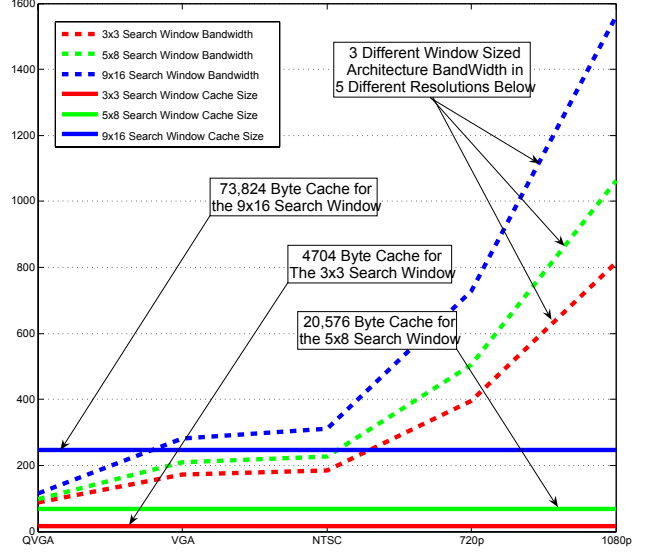


Figure 7: Cache-Min Algorithm Analysis

height of Search Window.

$$\begin{aligned}
 B_{sys} &= C + I + O + H \\
 &= C + I + O + H_{in} + H_{inter} + H_{intra} + H_{out} + H_{ref} \\
 &= C + 1X + 0 + 1X + rX + 0 + 0 + 1X \\
 &= C + (3 + r)X_{spd} \\
 C_{cost} &= C_{inter} + C_{intra} \\
 &= ((n - 1)lines of MBs + 1line Pixels + 16Pixels) \\
 &\approx (16(n - 1) + 1)Lines of Pixels
 \end{aligned} \tag{20}$$

Bandwidth & Cache Cost in "Bandwidth-Min" Approach

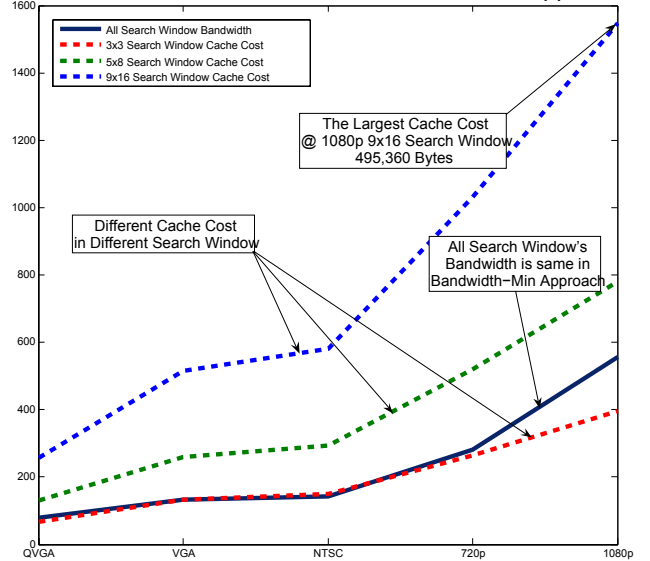


Figure 8: Bandwidth-Min Algorithm Analysis

4.3 Comparison of different algorithms

As shown in Figure-7 and Figure-8, both figures have the same scale in bandwidth and cache Cost. The X-Axis of the figure is the resolution level, which has been defined in Table-2. The Cache-Min approach gives us a model that the cache cost will not be modified by the increase of resolution. The cache is only related to the Search Window Size we have set. Hence, we are able to design a specific hardware with a cache limitation. However, as you can see in Figure-7, the bandwidth will go extremely high if the application resolution becomes 1920 by 1080. As the solution with low bandwidth constraint in some critical use (for example, low power application), we suggest the Bandwidth-Min approach as shown in Figure-8. The bandwidth in this approach will be less than 600 MHz. Likewise, the bandwidth becomes 1600 MHz in Cache-Min approach as shown in Figure-7. It means that the resolution is 1920x1080 in display with 16x9 Search Window used to refine the output quality.

Basically, hardware solution is the best solution for H.264 design, even though 1600 MB/s or 500 KB of memory is used under the tolerance of today's hardware limitation. However, things become more complicated when we put our chips into the real world. We have to use appropriate approaches to fit different requests of our system. Some may adopt the approaches to refine the hardware or algorithms, and others may refine the bandwidth and cache cost algorithms to match with system requests. But those considerations are back to the hardware part, you can even save more your bandwidth by increasing your cache memory. In some application like FULL-HD camcorder, it might be necessary.

5. CONCLUSION

In this paper, we have decomposed the H.264 bandwidth model into several components based on H.264 algorithms, and proposed two optimized implementation methods in bandwidth and cache cost analysis. We provide an analytical method because what we believe is that every single design is tailored to meet specific requirements which we are unable to predict precisely. With the proposed approaches, others can generate and evaluate their models simply in their early prototyping stage. In the case of critical bandwidth cost analysis, the proposed method will help our designers/architects to avoid unnecessary cost for potential of system re-design due to failure of the prototyping.

6. REFERENCES

- [1] J. V. T. of ITU-T and I. J. 1, "Draft itu-t recommendation and final draft international standard of joint video specification (itu-t rec. h.264 iso/iec 14496-10 avc)," *Document JVT-GO50*, December 2003.
- [2] I. E. G. Richardson, "H.264 and mpeg-4 video compression," pp. 27–28, August 2003.
- [3] C.-J. L. Tung-Chien Chen and L.-G. Chen, "Hardware architecture design of an h.264/avc video codec," *IEEE 2006*, August 2006.
- [4] A. T. H.-Y. C. Tourapis, "Fast motion estimation within the h.264 codec," *Conf. Multimedia and Expo, ICME '03*, vol. 3, pp. III – 517–20, July 2003.
- [5] S. M. He Zhihai, "A unified rate-distortion analysis framework for transform coding," *Circuits and Systems for Video Technology*, vol. 11, December 2001.
- [6] Y. H. T. C. T. W. L. C. C.Y. Chen, S.Y. Chien, "Analysis and architecture design of variable block-size motion estimation for h.264/avc," *IEEE Transactions on Circuits and Systems I*, vol. 53, pp. 578–593, March 2006.
- [7] M. T. S. K. M. Yang and L. Wu, "a family of vlsi designs for the motion compensation block-matching algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1317–1325, October 1989.
- [8] H. Yeo and Y. H. Hu, "a novel modular systolic array architecture for full-search block matching motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 407–416, October 1995.
- [9] Y. K. Lai and L. G. Chen, "a data-interlacing architecture with two-dimensional data-reuse for full-search block-matching algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, pp. 124–127, April 1998.
- [10] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1301–1308, October 1989.
- [11] L. D. Vos and M. Stegherr, "Parameterizable vlsi architectures for the full-search block-matching algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1309–1316, October 1989.
- [12] C. H. Hsieh and T. P. Lin, "Vlsi architecture for block-matching motion estimation algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 169–175, June 1992.