DESIGN AND IMPLEMENTATION OF A MULTITHREADED HIGH RESOLUTION MPEG4 DECODER ON SANDBLASTER DSP

Vaidyanathan Ramadurai, Sanjay Jinturkar, Mayan Moudgill, John Glossner
Sandbridge Technologies, 1 North Lexington Avenue, White Plains, NY 10601
vramadurai@sandbridgetech.com

ABSTRACT

In this paper, we describe the design, implementation and multithreading of a MPEG 4 decoder (simple profile) for high resolution (VGA 640x480) on Sandblaster DSP. The implementation is done entirely in C. Software solution provides high reusability, low cost and short development time when compared to dedicated hardware solutions. We describe the multithreading of time critical tasks that are processor intensive as well as memory intensive.

1. INTRODUCTION

MPEG4 is a multimedia standard adopted by the Moving Pictures Experts Group (MPEG) [1]. A simple profile MPEG4 decoder is defined by the standard for low complexity coding and decoding of rectangular video frames. Implementing an MPEG4 decoder on embedded processors has always been a challenging task. The computational and memory requirements for decoding high resolution video streams like VGA are demanding and require substantial optimizations. In this paper, we present the implementation of a simple profile MPEG4 decoder on Sandblaster DSP for high resolution VGA encoded video streams. We focus on multithreading and efficient handling of video frames using Direct Memory Access (DMA). We also describe the partitioning of various blocks in the decoder across multiple threads based on their computational and memory complexity. By exploiting these techniques, we achieve real time requirements.

This paper is organized as follows. In Section 2, we provide an overview of a simple profile MPEG4 decoder. In Section 3, we discuss the Sandblaster multithreaded processor. In Section 4, we describe the multithreading of MPEG4 decoder for high resolution VGA streams. Section 5 concludes this paper.

2. SIMPLE PROFILE MPEG4 DECODER

The block diagram of an MPEG4 decoder is shown in Fig. 1. A typical MPEG4 decoder unit consists of a Variable Length Decoder (VLD), Inverse Scanning, an Inverse AC/DC prediction, Inverse Quantizer and Inverse Transform (IDCT). In case of an inter coded block/frame, it is motion compensated before being sent to output. The reconstructed output frame is available in YUV format to be displayed. This procedure is applied for every macroblock and thus for every frame.

![Fig 1: MPEG4 Decoder Block](image_url)
3. SANDBLASTER DSP

Sandbridge Technologies has developed the Sandblaster architecture for a convergence device [3, 4]. The Sandblaster architecture supports the data types necessary for convergence devices including RISC control code, DSP, and Java.

As shown in Figure 2, the design includes a unique combination of modern techniques such as a SIMD Vector/DSP unit, a parallel reduction unit, and a RISC-based integer unit. Each processor core provides support for concurrent execution for up to eight threads of execution. All states may be saved from each individual thread and no special software support is required for interrupt processing. The machine is partitioned into a RISC-based control unit that fetches instructions from a set-associative instruction cache. Instruction space is conserved through the use of compounded instructions that are grouped into packets for execution.

The memory subsystem has been designed carefully to minimize power dissipation. The pipeline design in combination with the memory design ensures that all memories are single ported and yet the processor can sustain nearly 4 taps per cycle for a filter (the theoretical maximum) in every thread unit simultaneously. A RISC-based execution unit, depicted in the center of Figure 1, assists in every thread unit simultaneously. A RISC-based integer unit. Each processor core provides support for branch bounds to be computed and addresses to be efficiently generated. Intensive loop processing is performed in the SIMD/Vector unit depicted on the right side of Figure 1. Each cycle, a 4x16-bit vector may be loaded into the register file while two vectors are being multiplied, saturated, reduced (e.g. summed), and saturated again. The branch bound may also be computed and the instruction looped on itself until the entire vector is processed. This may be specified in as little as 64-bits.

To enable signal processing in software, the processor supports many levels of parallelism. Thread-level parallelism is supported by providing hardware support for up to 8 independent programs to be simultaneously active on a single Sandblaster core. This minimizes the latency in physical layer processing. Since many algorithms have stringent requirements on response time, multithreading is an integral technique in reducing latencies. The data-level parallelism (SIMD) is supported through the use of a Vector unit.

4. MULTITHREADING MPEG4 DECODER

The decoding routines are classified into parsing intensive functions like the VLD which is inherently serial and compute intensive functions like the IDCT and motion compensation. We also subdivide the compute intensive functions into memory-intensive (MI) and non-memory intensive (NMI) functions. For e.g. motion compensation (MC) is memory-intensive as it operates on current frame/pixels and reference frame/pixels. IDCT and Dequantize are non-memory intensive functions as they operate on smaller blocks and do not require reference pixels from memory.

Memory access becomes intensive depending on the type or level of memory being accessed. The memory system is divided into 3 levels on the Sandblaster DSP based on the speed and access time. First is the external memory which is the slowest and resides outside the chip. The second is the L2 memory, which is on-chip and faster than the external memory. On Sandblaster DSP, every core or 8 threads share 256kbytes of L2 memory. The third is the L1 memory which is also on-chip and faster than the L2 memory. Typically L1 memory runs at the speed of the DSP. Every DSP core has 64kbytes of L1 memory shared by all the 8 threads. The output frames and thus reference frames are stored in external memory or slow memory. Hence accessing these memory regions is extremely expensive. By dividing the decoding blocks based on their memory access pattern, it is easier to partition and load balance them across multiple threads. Hence memory intensive blocks like MC which require access to external memory are assigned to MI threads and similarly for the NMI blocks.

Table 1 shows the computational complexity of the simple profile MPEG4 decoder executed on a VGA (640x480) resolution clip encoded at 1mbps, 30 fps. Based on these distributions, appropriate number of threads was selected to
balance the load and maintain real time requirements. Every Sandblaster thread processing unit (TPU) runs at 75MHz.

<table>
<thead>
<tr>
<th>Function</th>
<th>Computation (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Length Decode</td>
<td>15%</td>
</tr>
<tr>
<td>Inverse Scan, Inv AC/DC</td>
<td>10%</td>
</tr>
<tr>
<td>Dequantize</td>
<td>15%</td>
</tr>
<tr>
<td>IDCT</td>
<td>20%</td>
</tr>
<tr>
<td>Motion Compensation</td>
<td>40%</td>
</tr>
</tbody>
</table>

Table 1: MPEG4 Computational Complexity

The macroblock control thread executes control functions such as whether the macroblock to be decoded is an intra-coded or an inter-coded macroblock and then invokes the appropriate decoding routines.

There are 10 threads used for decoding a macroblock. These are subdivided into 6 threads performing NMI routines and 4 threads for MI routines. One thread from the 4 MI threads also controls the output DMA for transferring the decoded frames to external memory and the input DMA for transferring in portions of the reference frames from external memory to L2 memory.

The 6 NMI threads work on luma and chroma decoding, one thread for every 8x8 luma block (thus, 4 threads are used here) and one thread for every 8x8 chroma block (thus, 2 threads are used here). The data used by these threads are all stored in fast memory. Figure 4 shows the blocks executed by each of these threads. Each thread performs Inverse Scanning, Inverse AC/DC prediction, Dequantization for inter-coded macroblocks and IDCT for inter-coded macroblocks. In case of inter-coded macroblocks, every thread runs independent of each other without any synchronization. In case of intra-coded macroblock, the luma threads and chroma threads run independently and completely in parallel. The output from the NMI threads is then handed over to the MI threads while the NMI threads decode the next macroblock.

Figure 3 shows the high level architecture for multithreading the decoder. The input encoded streams are stored in external or slow memory and are transferred to fast memory (L2) in smaller chunks. This L2 data is then processed by a VLD thread. The VLD thread performs all the parsing and bit level decoding for one macroblock. The output from the VLD thread contains information to decode the macro block (macro block type, motion vectors, coded coefficients etc). This information is then transferred to a macroblock control thread while the VLD thread continues to decode the next macroblock. The context shared between VLD and macroblock control thread is double buffered.

Figure 5 shows the blocks executed by MI threads. In case of inter-coded macroblock, every thread performs motion compensation on its 8x8 block and stores the output in L2 memory. The intra dequant and IDCT were moved to the MI threads just to balance the load on all the 12 threads.
The buffer management is also done by the MI threads. One thread checks if a single row of macroblocks (in case of VGA, this is 40 macroblocks) has been decoded. Once a row of macroblocks has been decoded to L2 memory, the output DMA engine is started to transfer the decoded pixels to external memory, while the next row of macroblocks is decoded to the second buffer in L2 memory. In this way, the latency of transferring the decoded pixels to external memory is very well hidden.

On the input side, the buffer management is implemented more like a cache. For e.g. a slice or a portion of the reference frame is DMA transferred from external memory to L2 memory based on demand. In our current implementation, we break the reference frame into 3 equal pieces, R1, R2 and R3 of N macroblocks each. Once the last macroblock of a frame is decoded, R1 of the reference frame is transferred using a DMA from external to L2 memory. By the time the motion compensation stage is reached, these reference pixels are available in L2 memory. Depending on the motion vector, a check is made if the reference pixels are in the L2 cache or in external memory. If there is a hit, the pixels are taken from L2 memory, otherwise from external memory. Similarly R2 and R3 are DMA transferred after decoding N macroblocks.

We have observed an 80% cache hit performance for most of the teststreams that were experimented with. Also, significant portion of cache misses were from boundary pixels that depended on reference pixels from the next or previous slice. This can be further improved by storing boundary pixels from the previous and the next reference frame slices.

With our implementation, we have achieved 30fps for VGA resolution, encoded at 30fps with an average bit rate of 1mbps. Even though 12 threads were allocated to do the processing, there is sufficient CPU availability within these threads to do other processing. This ensures that the current implementation can be easily scaled to higher resolutions like D1 (720x576).

5. CONCLUSION

In this paper, we have described the multithreading of a simple profile MPEG4 decoder on Sandblaster DSP for high resolution video streams like VGA. By exploiting thread level parallelism and using memory management techniques like DMA for frame buffer handling, we achieve real time performance requirements.

6. REFERENCES