Early Exploration of MIPS Cost and Memory Cost Trade-off for MediaDSP Media Processor

Johan Eilert, Dake Liu
Dept. of Electrical Engineering
Linköpings Universitet
S-581 83 Linköping, Sweden
Email: {je,dake}@isy.liu.se

***Abstract*— This paper presents an attempt at an early investigation or exploration of the trade-off between MIPS cost and memory cost. The cost model is an approximation based on knowledge of the application behavior and algorithm inner loop costs for the hardware architecture, which consists of a control processor and a SIMD processor. The application is an H.264 video decoder. Various memory configurations such as double buffer and other trade-offs between MIPS cost and memory size are compared.**

I. Background

Mobile platforms, such as mobile phones, are getting increasingly powerful in terms of memory capacity and computational power. This opens up new possiblities for new kinds of services.

Increased performance usually implies higher power consumption with more frequent battery recharges that degrades the end user experience. Therefore it is important to develop energy efficient solutions with enough performance and functionality.

Many types of video services such as video calls and television broadcast reception has emerged. These services are based on various international standards and it is interesting to use only one hardware solution for all standards.

The purpose of this work has been to make an early estimation of the (on-chip) memory requirements and how memory size can be traded for computational power. Memory size is an important issue as memories are typically the largest contributor to chip area and power consumption. Early estimations are important to get hints early in the design process before things get too difficult to change.

The rest of this paper is organized as follows: In the next section, the application is described, followed by a description of the hardware architecture. After the background information, the main work and the results are presented.

A. The H.264 Video Codec Application

The H.264 standard [1] for coding of video frames is used for delivering video contents to mobile platforms because it offers high video quality at modest bit rates. This section describes the basic principles of the standard.

The standard describes how a stream of video frames can be represented as a storage-efficient bit stream, and how to decode (restore) the video frames for playback or editing. Encoding is not explicitly covered, any encoder capable of generating a bit stream that can be decoded according to the standard is a valid encoder.

Initially, each original video frame is separated into luma (gray) and chroma (color) components and these are further divided into small rectangular macroblocks. Each macroblock is 16×16 pixels and it can be subdivided further, down to 16 blocks of 4×4 pixels. The macroblocks are coded individually, except that a macroblock can use prediction data from previously decoded macroblocks in the same frame (spatial prediction, a.k.a. intra-prediction), or from earlier frames (temporal prediction, a.k.a. inter-prediction).

Spatial prediction uses pixel values on the edges of neighboring blocks to cover the current macroblock or subblock in different ways.

Temporal prediction uses a small part from an older frame as prediction data for the current macroblock or subblock. The location of the source part in the other frame is calculated from the location of the current macroblock in the current frame, plus a motion vector. The calculated location is at quarter pixel resolution, and a 6-tap interpolation filter is used to compute the intermediary pixels.

The prediction is corrected with a residue signal. The residue is dequantized and the inverse transform is computed. The residue is added to the predicted data.

For low bit rates, there can be disturbing blocky artifacts caused by sharp transitions between the blocks in a frame. A final low pass deblocking filter is applied that selectively filters the block edges based on pixel values and several bit stream parameters.

Finally, the frame is ready and it is put away for later access, either as prediction data or for display. Frames do not have to be decoded in playback order, and this enables temporal prediction from "future" frames.

B. The MediaDSP Architecture

The architecture was designed specifically for H.264 and other video standards. It consists of four main parts as shown in Fig. 1. There are two processors, called Spock and Schubert, and a DMA controller. There is also an external video frame memory, it is located off-chip because of its size. Spock and Schubert are very different from each other as described below.

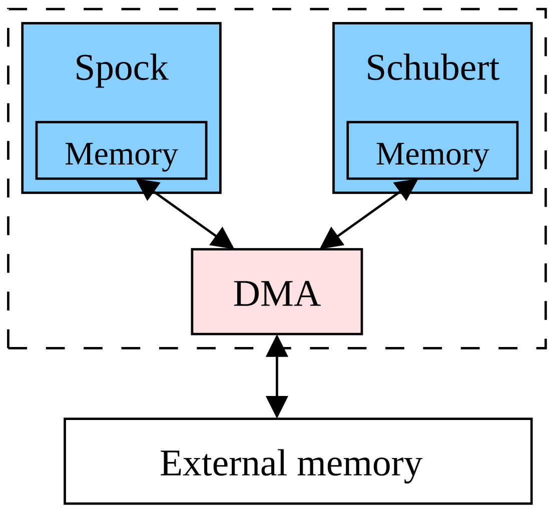

Fig. 1. Simplified view of the MediaDSP architecture.

Spock is a simple single-issue DSP. It is the global coordinator in the system. It is also responsible for scheduling the other resources, and for creating DMA command lists for the DMA controller in order to move data between itself, Schubert, and the external memory.

It has special hardware for accelerating bit-level manipulation such as variable length decoding (e.g., Huffman decoding).

Schubert is a SIMD processor. It has four 16-bit wide data paths and its main intended function is to do pixel level computations. Schubert can access up to four consecutive 16-bit values in a row or a column per clock cycle in its data memory. Each memory word stores two 8-bit pixels or one 16-bit intermediate value.

The data path in Schubert can perform both four parallel computations that generates four result as well as reduction computations that generates a single result from up to eight data inputs.

C. Mapping the Algorithm to the Architure

Since Spock and Schubert are specialized for different types of tasks, the mapping of H.264 to the architecture is quite straightforward.

Spock will do variable length (e.g., Huffman) decoding and other bit stream level computations.

Schubert will do all pixel-level computations such as dequantization, inverse transform, interpolation, and deblocking. All computations are performed on the smallest picture unit, a 4×4 pixel block, but the performance figures in this paper are shown for macroblocks.

II. The Work

A. Approximations

Some features in H.264 were ignored during this investigation. The most important parts that were left out are interlaced video, arbitrary slice ordering, and color processing. It is our belief that the first two are not very commonly used for mobile systems, and color processing can be approximated by multiplying all figures by the appropriate factor (e.g., 1.5 for 4:2:0 chroma subsampling).

It is assumed that all frames are P-frames, that is, they need to access one old frames for prediction.

The investigation covers Schubert's memory and MIPS cost only. Spock is assumed to be always capable of decoding the bit stream and generating the proper DMA lists and command lists to Schubert.

B. Memory Size

The first parameter that was studied is the number of macroblocks that Schubert can hold in its data memory. During the first part of the computation, that is, everything except deblocking, each macroblock needs source data such as its corresponding prediction data and residue data. The prediction data also needs a small border of surrounding pixels due to the 6-tap filter that is used.

Memory size of one, two, four, eight, and sixteen macroblocks and their source data has been considered in the investigation. The memory size does not affect the total amount of pixel work that Schubert must perform, but it does change how much data can be processed at a time, which affects the number of task initializations.

C. Double Buffering

Schubert memory can be double buffered to allow simultaneous Schubert calculations and DMA transfers. In a double buffered system, there are actually two memories. One memory is connected to Schubert and the other memory is connected to the DMA controller. When both Schubert and the DMA transfers are finished, the memories switch owners so that Schubert gets new data to work with while the DMA can transfer newly calculated results to the external memory and then fill the memory with new unprocessed data.

Not using double buffering makes all memory available for Schubert, but Schubert now also has to wait while the DMA transfers take place. Also, data that is needed in the next iteration can remain in the memory. This is not the case for double buffering since the memory will not be accessable by Schubert in the next iteration, and the needed data will have to be loaded from external memory again.

D. Optimized Loading

During the prediction step, data from a previously decoded frame is needed. This data can be loaded in a memory efficient way, which costs more for Schubert to access in terms of address calculations, or in a more memory inefficient way that enables cheaper address calculations.

The less memory efficient method loads a 9×9 block for each predicted 4×4 block. The extra pixels are needed for the 6-tap interpolation filter.

Operation	**Cost/MB**	**Setup cost**
Dequantization, inverse transform, interpolation	1954 cycles	26 cycles
Address calculations	80 cycles	0 cycles
Deblocking	1760 cycles	10 cycles
DMA transfer	64 cycles	10 cycles

TABLE I

CYCLE COST PER MACROBLOCK FOR VARIOUS OPERATIONS.

The memory optimized method assumes that most of the extra pixels can be shared by neighboring blocks, and it can thus load fewer pixels in total. It loads 21×21 pixels for one macroblock, which includes the border for the interpolation filter.

E. Cost Model

The costs for various operations by Schubert and the DMA controller are given in Table I. The setup cost is paid once for each iteration, and the total setup cost will decrease as the memory size increases since a larger memory allows more useful work per iteration. The address calculation part is only included when the optimized load option is used for prediction data.

The memory size and transaction sizes are given by the memory configuration: the number of macroblocks, whether double buffering is used or not, and whether optimized load of prediction data is used or not.

A single macroblock occupies $16 \times 16 = 256$ bytes, but the prediction data and the deblocking data need extra border pixels as described earlier, and the residue data also needs to be included. In the case with room for one macroblock in the memory, the memory requirement is approximately 800 bytes.

III. RESULTS AND FUTURE WORK

A small computer program was written for calculating the expected MIPS cost for decoding a CIF resolution (352×288) video stream with 30 frames per second, using various memory configurations, and with the restrictions and approximations mentioned above. The result is presented in Fig. 2 which shows MIPS cost vs. memory cost for all combinations listed above.

The results show that the single memory configurations will spend more MIPS than the double buffered configuration. This is not surprising since Schubert is wasting cycles waiting for the DMA transactions to finish.

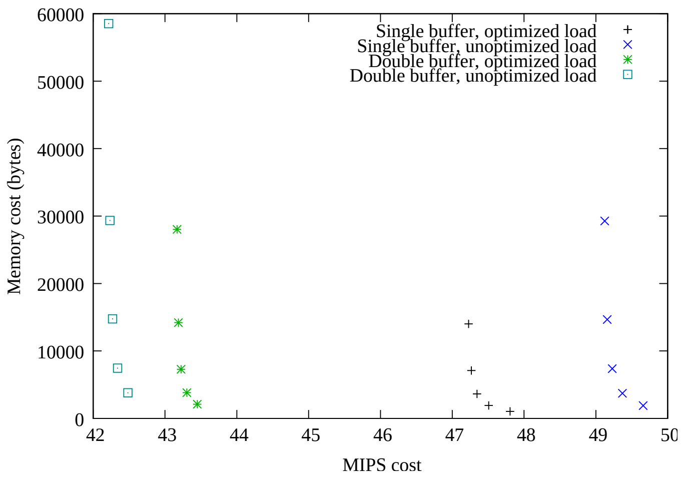

Fig. 2. MIPS cost vs. memory cost for various memory configurations. The bottom-most marker in each category shows the situation for one macroblock, the next marker above it is for two MBs, then four, eight, and sixteen MBs closest to the top.

For the double buffered configurations, the one with optimized load consumes more cycles because Schubert has to spend more time with address calculations. This is the opposite result from the single memory configuration where it is faster to load less data.

In all cases, the MIPS cost goes down when the number of macroblocks is increased because the overhead from initialization is reduced.

The results show that the MIPS cost does not vary that much with memory size, as long as double buffering is used.

As more details are known about the system, the cost model can be updated and the results will be more accurate. Eventually, the cost model should include Spock and the DMA controller.

REFERENCES

[1] ITU-T recommendation H.264, “Advanced video coding for generic audiovisual services”