INSTRUCTION AND HARDWARE ACCELERATION FOR MP-MLQ IN G.723.1

Mikael Olausson, Dake Liu

Computer Engineering
Department of Electrical Engineering
Linköpings University
SE-581 83 Linköping, Sweden

ABSTRACT

This paper describes a significant improvement in complexity for the higher bit rate, 6.3 kbit/s, speech coding algorithm G.723.1. The solution is to reduce the number of multiplication of the most computing extensive part of the algorithm. This part stands for around 50% of the total complexity. This is done by identifying and excluding multiplication with zeros. G.723.1 is one of the proposed speech coders in the H.323 standard. The work has been done by thoroughly examining the fixed point source code from ITU, International Telecommunication Unions [1]. A hardware structure for an application specific instruction set processor (ASIP) is proposed to increase the performance.

1. INTRODUCTION

The market for voice over Internet protocol, also called VoIP, has increased over the years. Voice has been a natural choice of communicating for a long time and will continue to be so. The H.323 standard contains five different speech coders with different complexities and bit rates. The first one is G.711, which is mandatory and uses A/u-law compression at 64 kbit/s. Another coder is the G.728 Adaptive differential PCM (ADPCM) at 16 kbit/s. The third is G.722 and works on the bit rates of 48/56/64 kbit/s. The last two are more interesting if we are dealing with bandwidth limited transmission channels. These are G.723.1 and G.729. While the first one have two different bit rates specified, 6.3 and 5.3 kbit/s, the last have three different, 6.4/8.0/11.8 kbit/s. These two both have parts that are common, but also parts that differ a lot. From a market point of view it is of highest interest to make the implementations of these algorithms as efficient as possible. A couple of factors may influence the choice of algorithm. For example some users want to squeeze as many channels as possible on a limited transmission channel. Then their choice is as low bit rate as possible if the speech quality is good enough. Others might use them in battery powered applications and their aim is low power consumption by decreased complexity with reduced speech quality as a trade off. The examination is not done in deep, rather from a behavior approach where we are not bound to a certain hardware manufacture. The quality or the robustness will not be treated.

Thanks to Swedish Foundation for strategic Research (SFF) and the Technical Research and Research Education (TFF) for funding.

2. GENERAL DESCRIPTION OF G.723.1

The frame size of G.723.1 is 30 ms(240 samples). In addition to this, the algorithm uses a look ahead of 7.5 ms, this gives a total algorithmic delay of 37.5 ms. The first component of the algorithms is a high pass filter to get rid of undesired low frequency components. The short term analysis is based on 10th order linear prediction (LP). These coefficients are calculated for every subframe, 7.5 ms or 60 samples. The unquantized coefficients are then transferred to Linear Spectral Pairs (LSP) for the last subframe. These LSP are then quantized using a Predictive Split Vector Quantizer(PSVQ). The excitation parameters from both the fixed and the adaptive codebook are determined on subframe basis. The codec then uses an open loop approach to calculate the pitch delay. It is estimated every 15 ms(every second subframe) in the

G.723.1. This pitch value is then refined in the closed-loop pitch analysis. The closed-loop analysis is done for every subframe. The gains and the pitch delay are referred to as adaptive codebook parameters. The fixed codebook extraction is the most exhaustive part of the whole algorithm. In this codec there exist two different approaches. One, which is used in the lower bit rate 5.3 kbit/s and is called Algebraic-Code-Excited-Linear-Prediction, (ACELP). This ACELP places at most 4 non-zero pulses within a subframe. The positions are determined from the codebook. The second approach is to use Multi-Pulse Maximum Likelihood Quantization (MP-MLQ). In this case you have more freedom to place the pulses more individually and not based on an algebraic codebook. It is in this part of the algorithm our proposal fits in. Any improvement in this part, will give great effects on the performance, because this part alone stands for around 50% of the total execution time.

3. STATISTICS OF BASIC OPERATIONS

All the arithmetic and logic functions like add, sub, shift, multiply and so on are implemented with a standard C library. This makes it simply to do statistics over how many times different functions are used. Additional to this, the C code has been thoroughly examined and all the branch, loop and move functions have also been identified and classified. All these statistics over basic operations, branch, loop and move instructions give a good hint on where to find improvements on instruction and architecture level. A more detailed description can be found in [2].

4. FORMER WORK

In order to handle as many channels as possible within a fixed bandwidth, we want an algorithm with as low output bit rate as possible. The drawback of a low bit rate is that the complexity is increased in order to keep the quality high. In [2] there were three hardware structures proposed. They all contributed to a lower complexity. One of those proposals also hold for other speech coding algorithms like G.729 and also the lower bit rate of G.723.1. The first one merges the instructions 32-bit absolute value, 32-bit comparison and conditional data move of both a 32-bit data and the loop counter into one instruction. This instruction turned out to be really useful for the speech coding, while they all use the analysis-by synthesis approach [3]. It is rather a trial and error procedure than a deriving procedure for finding the best fitting signal out of many alternatives. In addition to this we also presented an address calculation scheme including offset addressing and absolute value calculation. These calculations were incorporated within the address generator unit instead of using the ordinary arithmetic unit. While the first one could give complexity savings from 9-15%, the last two could give savings on another 5% in the higher bit rate of the G.723.1 speech coder.

5. HARDWARE IMPROVEMENTS IN G.723.1 6.3 KBIT/S

Here we will look at another part of the MP-MLQ. This part includes a lot of multiplications, where it turns out that many of them are just multiplication by zero. Here we can see a great potential for savings. The C code for a step of the loop looks like the following:

```
for ( j = 0 ; j < 60 ; j ++ ){
   OccPos[j] = (Word16) 0 ;
}

for ( j = 0 ; j < Np(5 or 6) ; j ++ ){
   OccPos[Temp.Ploc[j]] = Temp.Pamp[j] ;
}
for ( l = 59 ; l >= 0 ; l – ) {
   Acc0 = (Word32) 0 ;

   for ( j = 0 ; j <= l ; j ++ ){
      Acc0 = L_mac( Acc0, OccPos[j], Imr[l-j] ) ;
   }
   Acc0 = L_shl( Acc0, (Word16) 2 ) ;
   OccPos[l] = extract_h( Acc0 ) ;
}
```

This nested loop will be entered 64 times in the worst case and the total number of multiplication-and-accumulations will then be:

$$64 * \sum_{l=0}^{59} \sum_{j=0}^{j=l} 1 = 64 * \frac{59 * 60}{2} = 113280 \qquad (1)$$

Out of these 113280 MAC-operations are only 21120 actually multiplication with operand OccPos[j] not zero. That means that over 90000 multiplications are wasted. Before the loop starts all the entries to variable OccPos are cleared. We only have 5 or 6 array indices, Temp.Ploc[i] as positions, which are between 0 and 59. We also know their corresponding values, Temp.Pamp[i] as amplitudes, which are non zero. There are six elements for even subframes and five for the odd ones out of 60 entries in OccPos, which are not zero. So, instead of forcing the inner loop to multiply over all OccPos values, even the zero valued ones, we just loop over the non-zero values. The new proposed C code for this nested loop will look like the following:

```
for ( l = 59 ; l >= 0 ; l – ) {
   Acc0 = (Word32) 0 ;
   for ( j = 0 ; j < Np(5 or 6) ; j ++ ){
      if ( (l-Temp.Ploc[j]) >= 0 )
         Acc0 = L_mac( Acc0,
            Temp.Pamp[j], Imr[l-Temp.Ploc[j]] ) ;
   }
   Acc0 = L_shl(Acc0, 2);
   OccPos2[l] = extract_h(Acc0);
   }
```

This nested loop will also be entered 64 times in the worst case and the total number of multiplications has decreased to 64*60*5.5=21120. The number of multiplications has actually decreased even more, because some of the multiplication will not occur due to the branch operation. By implementing this in software on a DSP, the performance estimations can become tricky. We have introduced a conditional branch within the loop. While it is such a short loop, the penalty from pipeline stalls might be high. Some DSP's include operations, which can do conditional operations. A good solution here would have been a conditional multiply-and-accumulate.

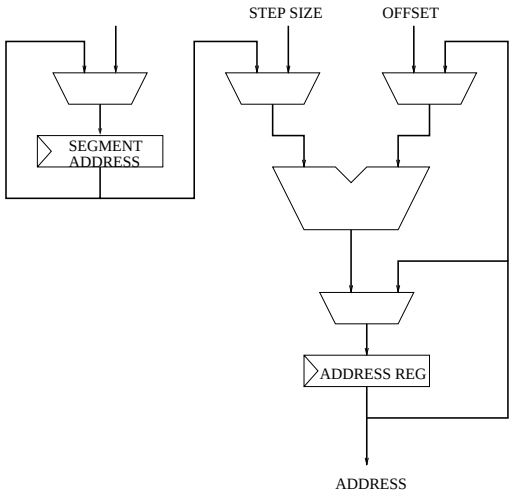

Fig. 1. Address generator with offset addressing and segment addressing.

To do this even better we will use the hardware architecture of figure 1, segmentation addressing. This one was originally presented in [2]. The offset calculation is rather simple, while it is a subtraction between the loop counter and the stored value of variable Temp.Ploc[j] in a register. The offset value is then added to the start address of the variable Imr. This modification is not enough, the calculated address offset, l-Temp.Ploc[i], can point outside the Imr buffer and we will perform an illegal multiplication. This can only occur when the result from the subtraction is negative. The solution is to use the msb bit from the subtraction between the loop counter and the Temp.Ploc[j] in figure 2. This data dependent control signal will then be propagated to the multiplier. If this bit turns out to be a one, i.e. a negative value of the subtraction result, the multiplier will not take the fetched operand as input, rather it will take a zero. We call this operation conditional operand. This will introduce an extra

multiplexer in the multiplier circuit. The reason for the extra absolute value calculation in the offset calculation circuit is found in [2]. We can address calculations, which require absolute value calculations.

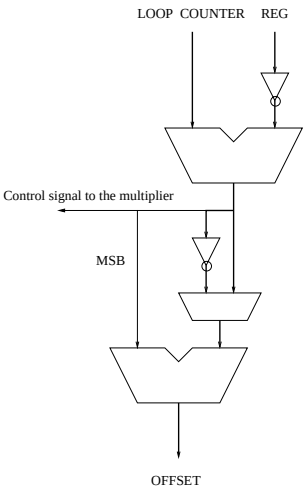

Fig. 2. Offset calculation with loop counter and absolute value operation.

6. PERFORMANCE ESTIMATIONS

In order to make an estimation on the performance we have weighted the operation by how many clock cycles they consume. This is of course very hardware dependent, but to get a rough estimate it is a good starting point. It is also important when evaluating the improvements. The table 1 below lists all the operations and their corresponding clock cycle consumption.

The branch instructions are weighted after the complexity in the comparison statement. If you compare with zero, then the cycle count is 1. For two 16-bit numbers the cycle count is 2 and finally, when you compare two 32-bit numbers the cycle count is 3. All the initialization of loops are counted as two clock cycles. When moving data are 16-bit movement treated as 1 clock cycle and 32-bit movement treated as two clock cycles. The only exception is when data is set to zero, then both 16-bit and 32-bit are treated as 1 clock cycle. Table 2 gives the estimated performance of the four different implementations. To make the estimate

Cycles	**Operation**
Arithmetic operations	
1	16-bit operations
2	32-bit operations
3	32-bit absolute value calculations
18	16-bit by 16-bit division
20	32-bit by 16-bit division
Branches	
1	comparison with zero
2	comparison between two 16-bit values
3	comparison between two 32-bit values
Loops	
2	Loop start
Moves	
1	16-bit move
2	32-bit move

Table 1. Number of clock cycles per operation.

even more accurate, we have also introduced memory related issues. We have taken into account that fetching operands from memories take one clock cycle, at least for the first operands of the loop. When you need the next operand of a buffer it is assumed to be in a register already. This fetch has been done during the operation of the operands. As we can see from table 2 there is no big difference between the branch implementation and the conditional multiply. This is not true because pipeline issues are hard to calculate. For a 2-stage pipeline, this is true, but for deeper pipelines you have to insert nop operations after the branch instruction. The total complexity of the whole algorithm is around 20 MIPS, without any modifications. This value is calculated by counting the basic operations and multiplying them by their weights from table 1.

7. CONCLUSION

In this paper we have seen four different implementations of a critical part of the fixed codebook search. By examining the code carefully, we have reduced the number of required multiplication by more than five times. This gives a performance improvement of about 30 % compared to the original implementation. While this number just corresponds to the fixed codebook ex-

Operation	Average MIPS	Maximum MIPS	Improvement
Original	8.40	11.08	
With branch	7.74	10.64	4-8 %
With conditional multiply	7.41	10.12	8-11 %
With extra HW	5.73	7.80	29-31 %

Table 2. Performance estimation for the four different implementations of the fixed codebook excitation for the higher bit rate of the speech coder G.723.1. The improvement in percent compared to the original implementation is stated in the fourth column.

citation part and this part stands for approximately 50 % of total execution time, the overall improvement is around 15 %. The modified hardware architecture to improve the performance is also presented. We have calculated the performance improvements by applying weights to all basic operations, add, sub, etc, branch statements, loop and move instructions.

8. ACKNOWLEDGMENTS

This work was financially supported by the Swedish Foundation for strategic Research (SFF) and the Technical Research and Research Education (TFF).

9. REFERENCES

[1] *ITU-T Recommendation G.723.1, Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s*, 1996.

[2] M. Olausson and D. Liu, "Instruction and hardware accelerations in G.723.1(6.3/5.3) and G.729," in *The 1st IEEE International Symposium on Signal Processing and Information Technology*, 2001, pp. 34–39.

[3] A.M. Kondoz, *Digital Speech*, John Wiley and Sons Ltd, 1994.