Optimizing a Software Packages
Multimedia Programming

Esad Akar
12/14/2017

Hardware details: AMD FX-8350 8 Cores, Windows 7, 4 gb ram

Run No.	Compiler	Total Encoding Time	Total ME Time	Speedup Relative to /O1 and /O2
1	Compiler baseline /O1	46.08	44.78	1
2	Compiler baseline /O2	33.54	32.47	1
3	Optimizations + /O1	28.96	27.59	O1: 1.60 O2: 1.16
4	Optimizations + /O2	25.08	24.02	O1: 1.84 O2: 1.34

Line Number(s)	File	Optimizations Used
405-486	Me_fullfast.c	Complete rewrote inner loop and replaced it with SIMD intrinsic.
592-683	Me_fullfast.c	Complete rewrote inner loop and replaced it with SIMD intrinsic.
845-90	Me_fullfast.c	Complete rewrote inner loop and replaced it with SIMD intrinsic.

Notes:
For the initial run with /O1 and no optimizations, the table below contains the run times.

SetupFastFullPelSearch	**19.888s**
iabs	10.219s
FastFullPelBlockMotionSearch	8.424s
computeBiPredSAD2	4.134s
HadamardSAD8x8	3.859s

SetupFastFullPelSearch took the longest time, and inside this function two inner loops near lines 405-486 and 592-683 had the highest impact. I replaced the first inner loop by packing 4 integers returned from iClip1, and subtracting the packed data from the packed srcptr values. The sum is accumulated in a separate xmm variable and the horizontal sum is calculated outside of the loop. This was done four times for each LineSadBlk variable.

In the loop near the lines between 592-and 683, the difference between refptr and srcptr are calculated. Like the steps from the first loop, both data streams get packed

and subtracted using intrinsics. The horizontal sums are performed outside of the loop once and assigned to block_sasd. These optimizations also removed the heavy use of iabs because `_mm_abs_epi32` was applied to the packed variables before summing up the differences in both loops. The run times were now:

FastFullPelBlockMotionSearch	**16.902s**
SetupFastFullPelSearch	8.307s
computeBiPredSAD2	3.628s
HadamardSAD8x8	2.789s
computeBiPredSAD1	1.790s

Notice that iabs no longer shows up on the list. Naturally, FastFullPelBlockMotionSearch was the next target. The loop in this function had the highest impact on run time, so all the optimizations were done there. The offset motion vector value never changes so this variable was packed fully on a 128 bit variable. Next, every iteration, 4 motion vectors from `p_Vid->spiral_qpel_search` is packed and the sum is calculated with the offset (the optimized code makes no calls to `add_MVs`). After these optimizations, the run times were now:

SetupFastFullPelSearch	**8.269s**
FastFullPelBlockMotionSearch	7.974s
computeBiPredSAD2	3.726s
HadamardSAD8x8	2.353s
computeBiPredSAD1	1.740s

I tried several OpenMp optimizations, but the complexity of the code and constant synchronization required slowed down the program substantially. One possibility is the encoder may benefit from having several frames being encoded in parallel but even in this case, the encoder works sequentially between frames and it needs certain previous frames computed first to move on to the next ones (I, B, P frame sequences).

Overall, the performance gains are substantial. The compiler also outputs even faster code after the optimizations were applied compared with the original /O2 unoptimized numbers.