DECODER COMPLEXITY AND PERFORMANCE COMPARISON OF MATCHING PURSUIT AND DCT-BASED MPEG-4 VIDEO CODECS

Ralph Neff, Toshio Nomura, Avideh Zakhor

Department of Electrical Engineering
University of California, Berkeley

ABSTRACT

Matching pursuits is an overcomplete expansion technique which has been successfully applied to the problem of coding motion residual images in a hybrid video coder. In this paper, the coding efficiency and decoder complexity of the method are compared to that of the DCT-based MPEG-4 standard with and without post-processing. Without post-processing, matching pursuits is shown to have significantly better PSNR and visual quality and similar decoding complexity compared to the MPEG-4 DCT decoder. To achieve reasonable quality at low bit rates, the DCT-based scheme requires post-processing, while the matching pursuit scheme does not. We show that the MPEG-4 post-processing filters have a prohibitive cost, increasing decoder complexity by a factor of 3 to 8. Finally, we introduce an all-integer matching pursuit implementation. Performance is shown to be within 0.05 dB of the original floating point algorithm.

1. INTRODUCTION

Most commercial and all standardized video compression systems are based on the motion-compensated DCT. At low bit rates, such systems show noticeable block edges and ringing artifacts. One solution is to introduce post-filtering to correct the artifacts. This is the method used by the DCT-based MPEG-4 Verification Model (VM), which includes optional de-blocking and de-ringing filters [3].

A disadvantage of postfiltering is that coded detail may be removed along with the unwanted artifacts. To counter this, decision criteria are used to adapt the filters to the strength of the artifacts. The decision process combined with the filtering adds a computational burden to the decoder, as will be seen later.

One way to avoid such problems is to use a basis set which does not introduce structured artifacts into the coded images. The matching pursuit video coder introduced in [1] does this by using an overcomplete set in which basis functions overlap arbitrarily in the coded image. Redundancy in the basis allows each transmitted element to closely match the original signal, and the structure of the basis set is not imposed on the coded images. Postprocessing is unnecessary and a high level of coded detail can be preserved.

Consider the class of applications in which video is encoded once and decoded many times. This is true for most internet video content, as well as video server and broadcast applications. Decoder complexity is critical in these cases, since the end user must decode and display the content in real time. Compression efficiency is also important, since the coded content must be stored and may be transmitted over low-bandwidth channels such as POTS. This paper will show that the matching pursuit encoder outperforms the post-processed DCT-based MPEG-4 system on these two points. However, it is important to note that the matching pursuit encoder complexity is higher than that of a hybrid-DCT encoder [1]. This makes it less appealing for applications which require real-time encoding.

The following section provides an overview of the matching pursuit video compression technique. Section 3 gives details on how the complexity counts were taken. Complexity and coding efficiency results are shown in Section 4, and a new integer matching pursuit implementation is presented in Section 5.

2. MATCHING PURSUIT

The matching pursuit coder used in this paper has been detailed in previous works [1, 2] and was recently adopted into the MPEG-4 Video VM [3]. The method decomposes motion residual signals f into basis functions called *atoms* over multiple stages. At each stage, a single basis function g_γ is chosen from the dictionary $\mathcal{G}$ in order to best represent the remaining signal energy in f. An energy pre-search identifies a local area in which to place the atom. The exact basis and weighting value p are then chosen by an exhaustive local inner product search. That is, each $g_\gamma \in \mathcal{G}$ is centered at each location (i, j) in the local region and the location and basis are chosen which maximize p:

$$p = < g_{(\gamma,i,j)}, R_n >$$

Here R_n is the remaining energy after n stages. Thus $R_0 = f$, and

$$R_{n+1} = R_n - p_n \, g_{(\gamma,i,j)_n}$$

The current-stage atom is thus subtracted from the working residual image and the remaining signal energy is passed on to the next atom search stage. After N atoms have been coded, the motion residual signal is approximated as:

$$\hat{f} = \sum_{n=1}^{N} p_n \, g_{(\gamma,i,j)_n}$$

The decoder reconstructs the motion residual by decoding the atom parameters and summing the atoms as above. This result is added to the motion prediction image to obtain the final reconstruction. The basis functions are of varying spatial extent and may exist at any integer pixel image location. Matching pursuit decoder computation thus depends on the number of coded atoms as well as their spatial extent.

3. MEASURING COMPLEXITY

The complexity of each decoder will be presented in terms of operation counts. Each addition, multiplication, division

The authors acknowledge support from ONR Grant N00014-92-J-1732, AASERT No. N00014-95-1-1163, AFOSR Grant F49620-96-1-0199, California State MICRO, Sun Microsystems, Philips, Tektronix, LG Electronics, and Sharp Corporation.

or bit-shift is counted as a single operation. Memory access and program control are ignored. The operation counts are thus theoretical, but are based on statistics from actual coding runs. The following subsections provide more details on how each part of the decoding time is counted.

3.1. Matching Pursuit Complexity

As described in Section 2, the complexity of the matching pursuit decoder depends on the number of coded atoms and their spatial extent. We use the basis dictionary from [1], consisting of the separable 2-D products of 20 1-D prototype functions. These functions range in extent from 1 to 35 pixels. To add a single atom into the reconstruction, each significant pixel must be weighted by the coded coefficient value and then added to the image. Thus two operations per pixel are required, with the number of pixels depending on the extent of the atom. This cost is summed over all coded atoms in the given frame.

3.2. Inverse DCT Complexity

The optimized IDCT algorithm presented in [4] is capable of performing a full 8×8 inverse DCT in 522 operations. Even better performance may be achieved for DCT blocks with few nonzero coefficients, since direct basis summation may be less expensive than a full IDCT algorithm. The basis summation method is equivalent to the method used in matching pursuit decoding, and we again assume that each 8×8 DCT basis requires two operations per pixel to incorporate the function into the image reconstruction. Thus the cheapest IDCT method is chosen for each block based on the number of nonzero coefficients, as shown in Table 1.

3.3. Motion Complexity

The same motion coding and decoding method is used for both the matching pursuit and DCT-based systems. Details can be found in [3]. Half-pixel interpolation is used to decode both luma and chroma motion blocks. In addition, an overlapping block motion model is used for luma. For our comparison, we count the number of operations needed to compute the luma and chroma prediction blocks.

3.4. DCT Post Processing

For a measure of the complexity of post-processing, we add up the operations needed for the de-blocking and de-ringing filters which are part of the MPEG-4 Video Verification Model 8.0 [3]. The count is again based on actual coding runs.

The de-blocking filter is applied first across the vertical and horizontal DCT block boundaries in both luma and chroma subblocks. For each filter location, a smoothness test is applied and one of two filtering methods is selected. If the data is judged to be part of a smooth image region, a strong smoothing filter is applied to four pixels on each side of the block boundary. If the data is not smooth, it is assumed that coded details are present and a weaker "default filter" is applied. This filter only affects one pixel on each side of the boundary. A further test is done to insure that any discontinuity removed at the border is smaller than the DCT quantization step, and thus could have been caused by quantization error.

Coefs (N)	IDCT Method	Complexity
0	None	0
1 to 4	Direct Sum	$N \times 128$
5 to 64	Full IDCT	522

Table 1: IDCT complexity as a function of the number of coefficients.

Table 2: Standard MPEG-4 Test Sequences

Seq	Name	Kbit/s	frame/s	size
1	Container	10	7.5	QCIF
2	Hall	10	7.5	QCIF
3	Mother	10	7.5	QCIF
4	Container	24	10	QCIF
5	Silent	24	10	QCIF
6	Mother	24	10	QCIF
7	Coast	48	10	QCIF
8	Foreman	48	10	QCIF
9	News	48	7.5	CIF
10	Coast	112	15	CIF
11	Foreman	112	15	CIF
12	News	112	15	CIF
13	Stefan	1024	30	SIF
14	Mobile	1024	30	SIF

The de-ringing filter is a 3×3 smoothing kernal applied adaptively after de-blocking. For each luma and chroma DCT block, a threshhold is computed which is normally the midpoint of the intensity range of the block. Each intensity value in the block is compared to this threshhold and a binary mask is generated; this mask contains a one if the corresponding pixel is above the threshhold, and a zero if the pixel is below threshhold. The filter kernal is then applied to only those pixels whose surrounding 3×3 region in the binary mask contains all zeros or all ones. A clipping rule is invoked so that no pixel may be changed by more than half the quantization stepsize.

4. RESULTS

In this section, the matching pursuit and DCT-based systems are evaluated based on decoder complexity and compression performance. We use the standard set of MPEG-4 test sequences, as shown in Table 2. Note that the leftmost column defines the sequence index used to label the remaining tables in this section.

Table 3 shows a breakdown of the complexity of the two decoders. The first two columns show the operation count per frame needed to invert the Matching Pursuit and DCT transforms, respectively. Note that the two inverse transforms take a similar amount of computation across the range of bit rates tested. The next column shows the complexity of decoding the motion prediction image. Motion decoding is much more complex than either transform inversion; it requires between four and seventy times the operation count of the IDCT, depending on sequence and bit rate. This is due to a combination of overlapping block motion compensation and half-pixel interpolation [3], each of which can be thought of as a filtering operation. It is thus expected that motion decoding dominates the decoder complexity for both Matching Pursuit and MPEG-4 systems in the absence of post-processing.

When post-processing is used for the MPEG-4 VM, the rightmost columns of Table 3 become relevant. These represent the complexity of the de-blocking and de-ringing filters specified in the MPEG-4 VM. These filters are a significant computational expense; when combined they require from three to seven times the computation of the motion decoder. Thus when the post-processing filters are included, they can be expected to dominate the computation of the MPEG-4 decoder.

Note that the complexity of inverse transforms IMP and IDCT generally increases with the bit rate, while the remaining columns show more dependence on image size. This is because Motion, De-block, and De-ring are all block based with a roughly constant complexity per image block. Thus each of these three columns shows similar per-frame complexity over the set of QCIF sequences (1-8), and much larger complexity for CIF and SIF sequences (9-14).

Table 3: Itemized Decoder Complexity (Ops per frame)

Seq	IMP	IDCT	Motion	De-block	De-ring
1	8705	11805	213391	856858	737079
2	8877	10081	202929	832574	698876
3	11744	8178	264768	874968	714539
4	18444	24798	204479	818782	725544
5	27003	19956	264265	721998	607235
6	25853	22685	257842	837196	704067
7	53336	49867	368900	762415	616044
8	60881	40136	396848	776557	671555
9	51287	42075	1005182	3719816	3139131
10	73109	31605	1495243	3675608	2936685
11	74723	20914	1498977	3850242	3047989
12	72723	68121	938580	3623375	3103931
13	351291	319043	1218875	2280845	1976359
14	175558	339340	1258895	1735694	1841017

Table 4: Total Decoder Complexity Per Frame

Seq	Millions of Ops			VM	(VM+PP)/
	MP	VM	VM+PP	%PP	MP
1	0.222	0.223	1.819	87.62	8.19
2	0.212	0.213	1.745	87.79	8.24
3	0.277	0.273	1.863	85.34	6.74
4	0.223	0.224	1.774	87.07	7.96
5	0.291	0.284	1.614	82.38	5.54
6	0.284	0.281	1.822	84.60	6.42
7	0.422	0.419	1.797	76.70	4.26
8	0.458	0.437	1.885	76.82	4.12
9	1.057	1.047	7.906	86.75	7.48
10	1.568	1.527	8.139	81.24	5.19
11	1.574	1.520	8.418	81.95	5.35
12	1.011	1.007	7.734	86.98	7.65
13	1.570	1.538	5.795	73.46	3.69
14	1.435	1.598	5.175	69.12	3.61

Table 4 shows operation count totals for the complete decoder systems. Note that the matching pursuit and DCT based MPEG-4 VM decoder have nearly identical complexity in the absence of post-processing. The column labelled "VM+PP" shows the complexity of the VM decoder with the de-blocking and de-ringing filters added. The post-processing takes between seventy and ninety percent of the total complexity, as shown in the column labelled "VM %PP." The final column shows the complexity ratio between postprocessed DCT and matching pursuits. The post-processed DCT is between three and eight times the complexity of the matching pursuit decoder. The operation count totals are shown graphically in Figure 1.

The numbers presented thus far count basic arithmetic operations in a way that is both machine and implementation independent. It is also important to consider how the algorithms run on a real machine. Table 5 shows profiling results using IPROF 0.3 [5]. Each table entry represents the number of machine instructions used to decode the standard test sequences on a 170 MHz Sparc Ultra-1. The MoMuSys implementation of the MPEG-4 VM is shown with and without the postprocessing filters. In this particular realization of the VM decoder, postfiltering accounts for between 40 and 50 percent of the real machine instructions, as shown in the last column. This result is smaller than the earlier prediction of 60 to 90 percent. One possible explanation is that the postfiltering routines are at a higher degree of code optimization than the motion and IDCT portions of the decoder.

To evaluate the systems based on compression efficiency, we provide both PSNR and visual results. The objective PSNR numbers are shown in Table 6. These are the same runs used to produce the complexity numbers presented earlier. Shown are luma PSNR results by sequence for the matching-pursuit based system and also the DCT-based MPEG-4 VM with and without post-filtering. The VM runs were produced using a constant quantization stepsize, a standard practice for MPEG-4 coding efficiency tests. The matching pursuit system was then run to match the bit

Figure 1: Comparison of total decoder complexity.

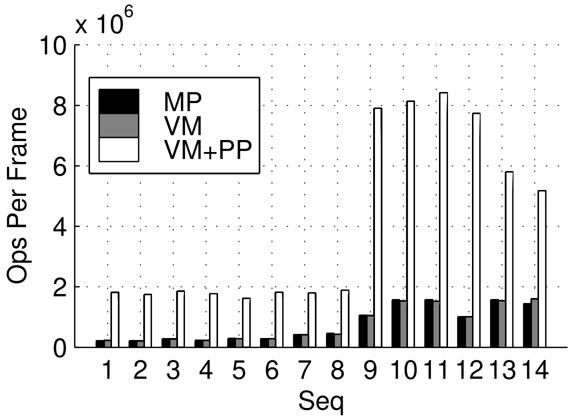


Table 5: IPROF analysis (millions of instructions)

Seq	VM	VM+PP	%PP
1	1795	3428	47.64
2	1687	3383	50.13
3	1852	3508	47.21
4	2493	4639	46.26
5	2500	4729	47.13
6	2576	4765	45.94
7	3005	4949	39.28
8	2979	4971	40.07
9	6859	13568	49.45
10	16306	28251	42.28
11	16060	28130	42.91
12	13700	27080	49.41
13	30190	53932	44.02
14	28950	49547	41.57

rate of the VM system for every frame. Two basic trends can be read from this table. First, the post-filtering has only a small effect on the PSNR of the VM, and it decreases the PSNR more often than it increases it. Second, the matching pursuit system produces significantly higher PSNR numbers than the DCT-based system, regardless of post-processing. This improvement is more than 1dB for half the tested sequences.

Alternately we can run the matching pursuit encoder to match the PSNR of the postfiltered VM run at every frame. For most sequences this results in a bit savings of ten to thirty percent for the same objective quality. This result is shown in Table 7.

The MPEG-4 post-processing filters have little effect on the PSNR, but a much greater visual effect. Figure 2 shows a sample frame of the Container Ship sequence coded at 10 kbit/s by the MPEG-4 VM without post-processing. Note the block artifacts which appear in the smoothly varying water. Additional distortion and ringing are seen in the small boat at the upper left of the image. Figure 3 shows the effect of the postfiltering. The block artifacts have mostly disappeared from the water, and the distortion around the small boat has been smoothed out. Now compare these images to Figure 4, which is the equivalent frame coded using matching pursuits. Both the water and the small boat are clean and free from systematic artifacts, despite the fact that no post-processing is used. Also note that the matching-pursuit image is sharper and preserves more detail than the post-processed DCT. This is seen mostly in the small boat and in the outline and features of the container ship.

5. INTEGER MP

Results presented above and in previous work [1, 2] are based on a floating point matching pursuit implementation. We now show that an all-integer version can be achieved

Figure 2: Container frame 50, coded by DCT-based VM.

Figure 3: Container frame 50, coded by DCT-based VM with post-processing.

Figure 4: Container frame 50, coded by matching pursuits.

Table 6: Comparison of Luma PSNR (dB). Matching pursuit matches bit rate of VM run.

Seq	MP	VM	VM+PP	Diff: MP-(VM+PP)
1	31.35	29.82	29.88	1.47
2	31.62	30.14	30.30	1.32
3	33.18	32.68	32.64	0.54
4	34.35	33.23	33.38	0.97
5	32.06	30.94	31.04	1.02
6	35.76	35.37	35.27	0.49
7	30.17	29.55	29.42	0.75
8	31.49	31.23	31.14	0.35
9	32.32	31.22	31.14	1.17
10	27.67	26.40	26.07	1.60
11	30.33	28.79	28.66	1.67
12	35.36	34.18	34.23	1.13
13	29.77	29.61	29.49	0.28
14	26.95	26.37	26.45	0.50

Table 7: Comparison of total bits. Matching pursuit matches PSNR of VM run.

Seq	MP-bits	VM-bits	% Saved
1	82569	103488	20.21
2	78657	101272	22.33
3	93815	105680	11.23
4	198334	261256	24.08
5	202451	253352	20.09
6	203037	235408	13.75
7	423292	515752	17.93
8	474274	495904	4.36
9	386228	479256	19.41
10	889317	1310512	32.14
11	1108129	1376664	19.51
12	874166	1146696	23.77
13	10225510	10640856	3.90
14	9050138	10198760	11.26

by adding a few scaling operations. The integer version is desirable because floating point implementations often differ slightly across platforms, causing decoder mismatch. An integer based codec may also utilize the integer SIMD instructions available on modern microprocessors to achieve significant speedups. Finally a floating point frame buffer is no longer used, reducing memory requirements.

The original floating point encoder uses 20 1-D elementary bases of varying lengths. The 400 separable combinations of these elementary bases form the 2-D basis library for coding residual images [1]. These 1-D bases have unit-norm in the floating point version. To represent these as integer filters, we introduce a scaling factor of $2^6 = 64$. Specifically,

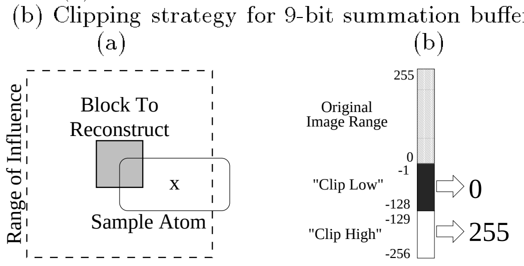
$$g_{int}(i) = RoundToInt(f_{float}(i) \times 2^6)$$

where g_{int} and f_{float} are the basis representations in the integer and floating point versions, respectively. Since each 2-D basis is the separable product of two scaled 1-D elementary bases, the 2-D basis is effectively scaled by 2^{12}, as are any computed coefficient values. During quantization, these scaled-integer coefficients are mapped to a set of integer bin indices, thus removing the scaling factor. Both integer and floating point MP versions thus transmit the quantized coefficients at the same scale. This is possible because each quantized coefficient is integer-valued, even in the original floating point version.

Each weighted basis function forms an atom, and the summation of coded atoms forms a representation for the residual signal. Decoding thus consists of adding all coded atoms to the motion prediction image. This atom summation process is done at the original image scale, i.e. the atoms are added directly to the 8-bit prediction image. Each atom-pixel value $p(i,j)$ is computed in a manner which maintains sufficient precision and simultaneously removes the scaling factors intrinsic to the basis vectors. That is,

$$p(i,j) = (g_h(i) \times ((g_v(j) \times p)/2^6))/2^6$$

Figure 5: (a) Blockwise atom summation for reduced buffer size. (b) Clipping strategy for 9-bit summation buffer.



where p is the quantized coefficient value, $g_h(i)$ and $g_v(j)$ are the scaled integer values of the horizontal and vertical 1-D basis components, and the downscale divisions by 2^6 are done using nearest-integer rounding.

The atoms are added to the prediction image in a summation buffer. We seek to minimize the size of this buffer, while maintaining an order independence property and insuring proper clipping. We will clarify these requirements and show that a 9-bit buffer is sufficient in practice.

Order independence means that the final reconstruction result is independent of the order in which the atoms are added to the summation buffer. This allows some useful decoder designs. For example, to minimize latency, the decoder should add atoms to the summation buffer in the order received. Alternately, one could rearrange the order of atom addition to reduce the spatial extent of the summation buffer. Figure 5a shows a blockwise design. The summation buffer is reduced to a block of arbitrary size, and all overlapping atoms are added to the block's motion prediction signal. A reduced summation buffer is desirable since the buffer must have higher precision than the 8-bits to which the final result will be clipped.

Higher precision and clipping are needed because of overflow and underflow. The original source image is 8-bits, as is the prediction image to which the atoms are added. The final summation result is the coded version of the 8-bit source. Since the coding is lossy, the final result as well as the intermediate summation values may exceed the [0,255] intensity range of the source. This is most common when source pixels are near the limits of the range. Clipping to [0,255] is thus necessary. However, we will avoid designs which clip the intermediate summation values, since this destroys the order independence property. For example, suppose a particular pixel has prediction value 250, and two overlapping atom pixels with values of +10 and -12. If intermediate values are clipped, the result clearly depends on the order of summation, as

$$Clip(250-12)+10 \neq Clip(250+10)-12$$

If only the final summation result is clipped, order independence follows simply from commutativity. This requires that the summation buffer be capable of containing the final result before clipping, as well as all intermediate summation results. However by defining the summation with modulo arithmetic we reduce the requirement such that only the final result must fit into the summation buffer. The proof of this is straightforward and follows from the fact that if a, b and c are integers with $a+b+c \in [0, M)$, then

$$(((a+b)_{modM})+c)_{modM} = (a+b+c)_{modM} = a+b+c$$

The summation buffer must then have sufficient depth to contain only the final result including any overflow or underflow. If we restrict the encoder such that the final summation result does not overflow the 8-bit range by more than 128 in either direction, then a 9-bit summation buffer with the clipping strategy shown in Figure 5b is sufficient. Using signed integer modulo-512 arithmetic, final values in the range [-128,-1] indicate underflow and are clipped to 0, while values in the range [-256,-129] indicate wrapped overflow and are clipped to 255. Table 8 shows that our proposed encoder restriction is already met in practice. For the range of tested sequences, the final summation result is always within 64 of the 8-bit range. We thus conclude that a 9-bit summation buffer allows proper clipping and preserves the order independence property. For a software implementation on a real machine, however, a 16-bit short integer implementation is more convenient.

Table 8: Maximum overflow in final summation result

Seq	Over	Under	Seq	Over	Under
1	40	-19	7	42	-24
2	36	-24	8	33	-17
3	0	0	9	46	-36
4	44	-25	10	47	-52
5	0	-16	11	40	-64
6	0	0	12	36	-30

Table 9: PSNR (dB) - Float vs. Integer MP

Seq	Float	Int	Diff	Seq	Float	Int	Diff
1	31.24	31.24	-0.01	7	29.98	29.96	-0.02
2	31.61	31.58	-0.03	8	30.96	30.91	-0.05
3	33.04	33.05	0.01	9	32.20	32.16	-0.04
4	34.29	34.28	-0.02	10	27.61	27.60	-0.01
5	31.95	31.94	0.00	11	30.10	30.08	-0.02
6	35.67	35.61	-0.05	12	35.28	35.24	-0.04

Table 9 shows the coding efficiency of the integer coder compared to the original float coder. The reduced precision of the integer coder causes only a slight drop in performance, within 0.05dB for all twelve tested sequences.

6. CONCLUSIONS

A matching pursuit video coder has been shown to outperform the emerging MPEG-4 standard in terms of both PSNR and visual quality. The matching pursuit system also has the advantage that it requires no postfiltering. This allows the decoder to remain simple, while avoiding the possibility that the postfilter removes coded details. The MPEG-4 VM uses postfiltering to remove DCT artifacts and thus obtains a visual improvement, though PSNR remains unchanged. The postfiltering has a prohibitive cost, however, increasing decoder complexity by a factor of three to eight. Simpler filters might be used, but there would likely be a tradeoff in visual performance.

A modified matching pursuit coder using integer math was shown to perform within 0.05dB of the original floating point version. Such a version allows reduced storage and potentially faster implementations on modern machines.

7. REFERENCES

[1] R. Neff and A. Zakhor, "Very Low Bit-Rate Video Coding Based on Matching Pursuits," IEEE CSVT, Vol.7, No.1, February 1997, pp.158-171.

[2] O. Al-Shaykh, E. Miloslavsky, T. Nomura, R. Neff and A. Zakhor, "Video Compression Using Matching Pursuits," IEEE CSVT, to appear.

[3] WG11 Document W1796, MPEG-4 Video Verification Model 8.0, July 1997.

[4] E. Feig and S. Winograd, "Fast algorithms for the discrete cosine transform," IEEE Trans. on Sig. Proc., September 1992, vol.40, no.9, pp.2174-2193.

[5] P. Kuhn, "IPROF 0.3: A highly portable instruction level profiler," available via anonymous ftp to ftp.lis.e-technik.tu-muenchen.de/pub/iprof