References

[1] Hin Soon Chin, John W. Goodge, Roy Griffiths, and David J. Parish. Statistics of video signals for viewphone-type pictures. *IEEE Journal on Selected Areas in Communication*, 7(5):826–832, June 1989.

[2] Andy C. Hung and Teresa H. Meng. Statistical inverse discrete transforms for image compression. *Proceedings of IS&T/SPIE Electronic Imaging*, 94.

[3] S. Keshav. Experiences with large videconferences on xunet ii. *Proceesings of INET*, June 1994.

[4] Margaret Rose Masrtonosi. Analyzing and tuning memory performance in sequential and parallel programs. Technical Report CSL-TR-94-602, Stanford University, 1994.

[5] M.Ghanbari. Two layer coding of video signals for vbr networks. *Journal of Selected Areas in Communication*, 7(5):771–781, 1989.

[6] Henry Moreton and Jeannine Smith. Hdcc - a software based compression algorithm for video conferencing, 1994.

[7] Ketan Patel, Brian C. Smith, and Lawrence A. Rowe. Performance of a software mpeg video decoder. *Proceedings of ACM Multimedia*, 1993.

[8] H. Schulzrinne and S.Casner. Rtp: A transport protocol for real-time applications. Internet Draft, draft-ietf-avt-rtp-04, October 1993.

[9] SGI. Indy technical report. Technical Report INDY-TR-06-93, Silicon Graphics, 1993.

[10] Willem Verbiest and Luc Pinnoo. A variable bit rate video codec for asynchronous transfer mode networks. *IEEE Journal on Selected Areas in Communication*, pages 761–770, June 1989.

[11] Gregory K. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, 34(4):31–44, April 1991.

Figure 10: Varying the Frame Rate

Figure 9: Variation With Cache and Line Size

fast. This is primarily because a large number of the misses are caused by data that is read once into the cache, at each frame interval. Thus increasing the size of the cache doesn't lead to improvement in performance once most of the other data structures fit into the cache.

Varying the inter-frame spacing on the sequences to study situations where we are not able to capture at 30 *fps* , was also done. We expect only the statistics for the block selection algorithm to change as we vary the inter frame spacing. The results are shown in figure 10. We plot the distribution of the mean squared error between the $(x, y)th$ block in the ith frame and the $i + 1th$ frame averaged over all x, y. The results are surprising, we would expect the shift of the curves to the right to be mcuh more than what is seen. A careful study of the actual sequences reveals the reason for this is that the motion is restricted to very small areas of the frame and the scene in most video conferencing situations remain constant.

6 Concluding Remarks

We have studied the performance of a software based video conferencing system. Each component of the system was analyzed for its computational, amemory and compression characteristics. It was found that over 35% of the time was spent due to cache misses, waiting for the video data to appear in the cache. We are able to reduce this time to 6-10 % of the total time. We were able to obtain better compression using the DCT algorithm with threshholding. The motion detection algorithm was shown to be robust. The decreased resolution between frames did not affect the characteristics of the system. We also showed that increasing the cache size did not bring large gains due to the nature of the cache misses. Increasing the line size helped to reduce the misses. Thus traditional architectures where the video data has to be brought into the processor cache are not a good solution. We must try to avoid the transfer of data from memory to cache in order to get better memory performance.

Figure 8: DCT with Thresholding

memory enhancements proposed above. It shows that we are able to obtain much better compression. The DCT is computationally more expensize as compared to the Haar transform.The computation is completely dominated by the transform stage. The slight increase in cache misses were tracked down to the kernels used by the inverse transform.

Figure 7 shows the characterisitcs for the individual stages for the DCT based system. Figure 7 (c) shows the probability distribution for the runs of zeros. As compared to 4(e) we see that the proability for larger runs is greater. The probability distribution for the bytes per block (figure 7 (d)) is also shifted to the left as compared to figure 4 (f).

Another surprising thing was since we are not doing any form of entropy coding as bit streams are costly to operate upon, using quantization with DCT does not improve the compression as well as expected. If we use thresholding with DCT instead of quantization we get the same oamount of compression figure. 8 but a much better picture quality. Thus it is much better to do thresholding and it is cheaper since it avoids divisions. [6]

5 Discussion and Realted Work

Figure 9 (a) shows the effect of varying the line size for the primary cache, while we keep the cache size fixed at 16K. It is important to note that we did not vary the miss cost depending upon the line size. We would expect the cost to increase with the line size. Thus depending upon how the cost scales it might be advantageous to increase the line size. The observed behaviour can be explained from the fact that most of the misses are incurred when the line is brought into the cache for block selection. As we increase the line size we incurr fewer misses and thus the improvement in performance.

Figure 9 (b) shows that the gains obtained by increase the size of the primary cache saturate very

[6]in the implementation of the dct stage we had optimized the quantization to use only shifts and adds. A couple of quantization matrices were hard coded in order to achieve efficency. Thus although the code segment size increased, the gain in speed was worth it.

Figure 7: DCT Compression Stages

Stage	Instruction %	Memory Stall %	Compression
Format Conversion	5.1	1.2	-
Block Selection	2.2	61.3	10.92
Transform	49.9	13.2	-
Thresholding Run Length Encoding	28.1	20.1	14.0
Absolute Counts	2.8 e+08	0.23-0.38 e+08	152.00

Table 2: NV with Memory Enhancements

Stage	Instruction %	Memory Stall %	Compression factor
Format Conversion	2.6	1.6	-
Block Selection	1.8	61.1	10.92
Transform	79.3	16.2	-
Quantization Run Length Encoding	10.5	16.3	21.0
Absolute Counts	6.1e+08	0.23-0.42 e+08	228.9

Table 3: NV with DCT and Memory Enhancements

Then cache misses were again incurred when data was copied from *current_y_data* into *transform_block* . It is also important to note that the cache misses in the *block selection* stage can't be avoided. In order to reduce the cache misses in the foramt conversion stage we do *block selection* on the frame buffer and make a single copy directly into *transform_block* for only those blocks that are transmitted, as shown if figure 6. The choice of the block selection algorithm is crucial. the algorithm used by NV uses only one line of a image at a time which is contiguous in memory. Thus we save on the miss cost for the blocks which are not sent. Since the number of transmitted blocks are 10% of the total this brings us tremendous savings in terms of cache misses. The statistics are shown in Table 2.

It can be seen from table 2 that the total memory stall time is reduced by a factor of 4-5 as compared to the base NV system. The format conversion stage is a null stage and does some buffer management. The misses are now incurred in the block selection stage. The relative distribution of the other misses remains almost the same. We also get a small reduction in the computation time, which otherwise remains the same. [5]

4.2 DCT based Transform Stage

We expiremented with another modification of the NV system by replacing the Haar based transform stage to a DCT based one. We expiremented with several algorithms for forward and inverse DCT. Finally we used the Jpeg DCT for the forward transformation stage. For the reverse transformation we used the DCT from [2] which exploits the fact that most of the coefficients are zero (this is similar to the optimizations done by [7]). Table 3 shows the performance of the DCT system with the

[5] On the lighter side, in our initial attempts to eliminate cache misses we had made local copies of both the uv and the y data. By coincidance these happened to be exactly 8K apart which is the size of the direct mapped cache and we noticed tremendous amount of miss rates as one was throwing the other out. Someday compilers will be intelligent enough !

Figure 6: The flow of data to Reduce Cache Misses

between blocks having a given *distance*. The *mse* remains low for blocks with *distance* less than 64 and the curve for the cumalative distribution also saturates around 64. If we choose a higher value we don't reduce the number of blocks sent but would increase the error due to the blocks not sent by a large amount.Thus 64 is the optimum choice for *CRT* .

Figure 4 (c) shows the distribution of the pixel values in the input data. It can be seen that the image energy is not compacted in one region and but is concentrated in a number of clusters, which represent spatial redundancy. 4 (d) shows the distribution of the transformed coefficients. The transformation compacts the energy into a few Haar coefficients.The transformation also reduces the entropy of the data.[4] (e) shows the probability distribution from the length of consecutive zeros (after thresholding).

Figure 4(e) shows the probability distribution ot the number of bytes sent per block. The probability of sending a few bytes is large although the average is dominated by the small percentage of blocks that give us less compression. Figure 4(f) shows the probability distribution for the runs of zero's before the runlength encoding stage.

Figure 5 shows similar statistics for the chrominance data. The only qualitative difference being that Figure 4 (c) shows that the probability of having large runs of zeros is very high.

4.1 Memory Enhancements

An analysis of the cache misses for the original code showed that most of the cache misses were incurred in the *format conversion* stage where the data was read from the *frame_buffer* into *current_y_data* .

of our test sequences here

[4]If we had maintained a very large precision then both would have the same information content, but since we use a fixed precision the entropy reduces.

Figure 5: Compression Stages - uv data

Figure 4: Compression Stages - y data

Stage	Instruction %	Memory Stall %	Compression factor
Format Conversion	36.3	77.1	-
Block Selection	2.1	5.1	10.92
Transform	32.0	15.2	-
Thresholding	14.1	1.4	14.0
Run Length Encoding			
Absoulte Counts	3.0 e+08	1.38-1.84 e+08	152

Table 1: Base Performance of NV

block. (A basic block is a region of the program that can be entered only at the beginning and exited only at the end).

The limitations of pixie being that it assumes that all instructions take one cycle and leaves us clueless about the memory behaviour. *Memspy* allows us to model only a one level direct mapped cache.

One of the difficulties in running instrumented code is that the instrumentation should not change the behaviour of the code. This is a problem with programs that base their behaviour on the system time elapsed. This was augmented in particular. Also the invalidations misses have to be artificially generated.

The sequences used tried to cover the range of applications that NV has been used for. Each sequence used was 750 frames and was captured at a resolution of 640 X 480. They included two talking head sequences, a sequence of a talk using overhead (which is the kind of video sent from IETF conferences). The other kind of sequences included sequences with split image of two speakers and a class where the camera is focussed on the blackboard and the writing pad.

4 Base Performance of NV

In this section we give the results for the computation and compression characteristics for the base NV system. Table 1 shows the statistics for NV. It can be seen that about 35% of the time was spent in the original code waiting for the data to come into the cache. The distribution of the misses among different stages shows that the format conversion that is done on the incoming image frame accounts for over 70% of the total misses. The other major components being the forward and the reverse transform which contribute 15-20 % .

Table 1 shows that format conversion takes a 36.3 % of the total computation time, which is comparable to what the transform stage takes. (It is important to point out that the publically available code is already optimized to use 32 bit operations wherever possible) . The compression achieved due to block selection is about 10.92 %. Which means that around 9the blocks are transmitted. The compression achieved in encoding each block is about 14. The overall compression ratio is thus about 150.

Figure 4 and 5 shows the outputs of various stages in the pipeline for a particular input sequence [3]. Figure 4 (a) shows the plot of the cumalative distribution of the number of blocks below a given *distance* in the block selection stage. reffig:yocs (b) shows the plot of mean squared error (*mse*)

[3] Although the probability distributions for the input data for various sequence look quite differnet the statistics for the other stages of the pipeline are similar and almost indistinguishable. Thus we choose one sequence that is representative

rate, the underlying architecture and a vague notion of quality.

For all the simulations a small set of video sequences were considered. The code was modified to read sequences from a file without affecting the flow of data throught the various data structures and the computation that was being performed. The interframe spacing could be adjusted.

When analyzing the compression characteristics we assumed the model that we are always able to do as much computation as we want to. So in this model we measured the various things by deciding the frame spacing, assuming that anything upto 30 frames per second could be processed.

When analyzing the memory behaviour we model the underlying memory system by a two level cache. The size of the primary cache was taken to be 16 Kbytes and the size of the secondary cache was 1 Mbyte. The miss latency for the primary cache was 14 cycles while that of the secondary cache was 70 cycles. We studied the effect of increasing the cache line size for both the primary and the secondary cache and also varying the size of the primary cache. Although we realize that on some systems the cache latency changes with changing the line size but we did not try to model that.

We tried to keep the quality across various simulations at a given frame rate constant. In the sense that all runs run on the same input and since they use the same block selection algorithm the same blocks are selected for transmission. Thus once we deicided a frame rate it fixed the quality for the transmission.

For each of the frame rates we measured the following quantities

- *Compression Achieved* The compression achieved was measured at the granularity of individual blocks and the frame as a whole.
- *Computation Reuirements* in terms of flops, load/stores per frame. The average number of instructions executed for each basic block were evaluated. By basic block we mean code with no branches. This was done at the procedural level only. So at the procedural level it gives the number of instructions at the average for each procedure invocation and the frequency of the procedure call.
- *Memory Behaviour* like cache misses and the miss costs depending on the architecture. The cache misses were calculated for each of the major data structures. Also the what data caused the cache misses to occur was also evaluated. The misses were classified into first reference misses, replacement misses and invalidation misses. For us invalidation misses occur only becuse of the video capture board DMA ing the frame into the image buffer. The distribution of the memory stall time across procedures was also considered.

To measure things like compression achieved we simply instrumented the code. Computational Requirements and memory behaviour were measured and modeled by *pixie* and *memspy* respectively.

Memspy [4] is a tool to help identify the memory bottlenecks in both sequential and parallel pograms. It provides information such as cache miss rates, cause of cache misses, and information on cache invalidations. It provides this information for both data, as well as code for the application. Memspy adds the code for information on the time spent in every procedure. To get information on a data structure level each data pointer that needs to be tracked has to be allocated through the Memspy runtime environment.

Pixie is an object instrumentation tool that can be used to measure code execution frequency in a program for performance analysis. Pixie reads an executable program, partitions it into basic blocks, and writes an equivalent program containing additional code that counts the execution of each basic

Figure 3: The flow of data

timestamp is greater than the timestamp of the current frame we assume that the frame has ended.

2.6 Data Structures

We present here the major data structures that are present in NV. These will be present in one form or another in almost all such software codecs. The video capture board captures a frame. It digitizes it and converts from the *rgb* domain to the *yuv* domain. The *yuv* frame is then stored in a buffer which is shared between the application and the video subsystem (figure 3). The format of the stored data is 4:2:2.

The *frame_buffer* is then scanned and the video data decimated in the x dimension from 640 to 320, also the y and uv components are separated and kept in two different data structures. The block selection algorithm is run on the *current_y_data* and also uses a copy of the previous frame's y data which is stored in the *old_y_data* . The *age* is also updated for each block.

For each block selected the image data is extracted from *current_y_data* into *transform_block* , which now holds the block data in contigous memory locations. This block is then transformed and *transformed_block* holds the coefficients of the transform, which then undergoes thresholding and runlength encoding to give us *encoded_data* .

3 Evaluation Methodology

We wanted to identify and isolate the effect of the major independent variables, which affect the performance of the system. The major variables that we identified were bandwidth available, frame

are not a general occurence

can be represented by the following equation where, $\overline{x}$ is the input vector and $\overline{z}$ is the transformed vector.

$$\overline{z} = \begin{pmatrix} 1/8 & 1/8 & 1/8 & 1/8 & 1/8 & 1/8 & 1/8 & 1/8 \\ 1/4 & 1/4 & 1/4 & 1/4 & -1/4 & -1/4 & -1/4 & -1/4 \\ 1/2 & 1/2 & -1/2 & -1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 & -1/2 & -1/2 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \overline{x}^t \tag{5}$$

An important thing to note here is that in calculation of the transform only additions and right shifts are required. Same is true for the inverse transform. This is important as on many machines multiplication is a really costly operation.

2.4 Thresholding and Run Length Encoding

In order to achieve compression, the transform coefficients are normally quantized. Thresholding being a restricted form of quantization. NV uses thresholding following which it does run length encoding. If we denote $T^{x,y}$ to be the transformation of the $(x, y)th$ block *i.e.* $F^{x,y}$.

$$\begin{aligned} T^{x,y}(i,j) &= 0 \quad if\ F^{x,y}(i,j) < threshold,\ x > 2,\ y > 2 \\ &= F^{x,y}(i,j) \quad otherwise \end{aligned}$$

The algorithm does not threshold the low frequency coefficients as the degradation in quality due to that is large. Also the threshold can be varied and from time to time blocks may be sent with a threshold value of zero *i.e.* losslessly encoded.

The thresholding phase is followed by runlength encoding. Here we try to code a run of zeros efficiently. The byte stream after runlength encoding has the following syntax. Each run is preceeded by a byte that describes it. We call this the *preamble* . The first two bits of the *preamble* tell us how many non-zero values are there in the run. The remaining bits tell us how many zero values are followed by these non-zero values. There are many different ways to code the runs, the choice is governed by the distribution of data.

2.5 Decoding and Displaying

The receiver of an encoded video stream carries out the reverse process as to what the encoder did. It expands the runlengths, and carries out the inverse transform. In order to display the image it has to convert from the *yuv* domain to the *rgb* domain. The display is updated onlt after all packets for the next frame have been received. If we wait till the end of frame to display the data there has to be an end of frame marker. The packetization used at the encoder in NV is RTP [8]. The RTP header which has an end of synchronization field, which is used to demarcate the end of a frame. In order to make the scheme robust against packet loss the timestamp is also used for the same purpose. All packets belonging to a frame have a single timestamp. Thus the algorithm for updating becomes, if we receive an end of frame marker we assume the frame has ended [2] or if we receive a packet whose

[2] here an implicit assumption is that packets are delivered in sequence. It is our experience atleast on the MBONE that a large number of out of sequence packets generally point ot problems in the network configuration or topology, and

$$D(F_{n-1}^{x,y},\ F_n^{x,y}) \;=\; \sum_{i=1}^{8}\sum_{j=1}^{8} |F_n^{x,y}(i,j) - F_{n-1}^{x,y}(i,j)| \tag{2}$$

Based on $d^{x,y}$ a decision is made, whether to transmit the block or not. If we decide that the *(x,y)th* block needs to be transmitted then we code $F_n^{x,y}$ which corresponds to *intraframe* coding. We have to choose a threshold CRT : conditional replinishment threshold such that

$$\begin{array}{lll} d^{x,y} & \leq & CRT \qquad \textit{do nothing} \\ d^{x,y} & \geq & CRT \qquad \textit{do intraframe coding} \end{array}$$

In practice instead of using $F_{n-1}^{x,y}$ in eqn. 2 we use $\overline{F}_{n-1}^{x,y}$, which is what a receiver would see after the block was encoded and then decoded. Typically there is a loss in quality so one would prefer to use this. Since the coded picture is highly compressed, motion detection in the pel domain (as in eqn 1) due to quantization noise might cause false motion detection.

The distance metric used by NV is the sum of the absolute difference on a single line of a block consisting of eight pixels. The line considered is not fixed but is rotated after each frame. Thus if one compares the 8th line in the block for frame i we will compare the 1st line in the block for frame $i+1$. NV runs a postfilter on the two dimensional distance metric calculated for all blocks in the image to smooth out the distances. Then whatever blocks have their distances above the threshold are transmitted. This can be expressed as

$$D^{'}(F_{n-1}^{x,y},\ F_n^{x,y}) \;=\; \sum_{j=1}^{8} (F_n^{x,y}(s,j) - F_{n-1}^{x,y}(s,j)) \quad ,\ where\ s = n\ mod\ 8 \tag{3}$$

$$D(x,y) \;=\; \max_{u=x-1,x+1}\ \max_{v=y-1,y+1} D^{'}(u,v) \tag{4}$$

When a block is sent its D value is set to zero. In block selection based algorithms there would be some blocks which don't change (for whom $d^{x,y}$ is always small). They have to be transmitted periodically in order to ensure that errors don't propagate and for example in a video conferencing system people who join the conference later do get the whole image. These blocks constitute the background. A fized proportion of the bandwidth is devoted to the transmission of background blocks. Thus for each block a number called the *age* is maintained. The *age* denotes number of frames that a block has not been sent for. The *age* of a block is used as a priority when sending background blocks. The *age* being decremented when the block is sent.

2.3 Block Transformation

The blocks that are selected for transmission are typically transformed to get compression. Given that a number of people may join a video conference,the algorithms are chosen such that multiple streams may be decompressed, and a single video stream maybe compressed in real time. This, of course, restricts the choice to a few simple transform algorithms.

NV uses the 2d Haar transform for compressing and decompressing the video data. The simplicity of the Haar transform makes it an excellent choice for software video compression. For instance, a 2D Haar Transform on an 8x8 block of data requires 224 integer operations. Although Haar transform is simple to implement, it's energy compaction properties are not as good as some other transforms. The two dimensional transform is done as two one dimensional transforms. Mathematically the transform

Figure 2: The Indy Video Subsystem

component of the codec that actually captures the frames and the rest of the code is fixed. The main interaction between these two parts is the format of the frame that is passed between them. The format expected by NV is of size 320x240. The Y component is represented in a raster scan order for the image. The UV components are interleaved. Since the format was 4:2:2 both the chrominance and luminance components are equal in size. It will be shown later that this interface becomes a bottleneck in processing of the video stream.

2.2 Block Selection Algorithm

In a conditional replinishment system, the image is segmented into small regions. The easiest and the most prevalent method is to break up the image into fixed size blocks (like 8x8). When a new frame is captured, for each block a number known as the *distance* is calculated. If we represent the frame as a two dimensional array of blocks where F_n denotes the *nth* frame ,$F_n^{x,y}$ denotes the (x,y) block in the *nth* frame and $F_n^{x,y}(i,j)$ denotes the (i,j) the pixel in the corresponding block, then the *distance* for block (x,y) $d^{x,y}$ is calculated as

$$d^{x,y} \;=\; D(F_{n-1}^{x,y},\; F_n^{x,y}) \tag{1}$$

The choices of the function $D(.,.)$ are mean squared difference or taking the difference for one line in the block or just the absolute sum of differences of some random pixels and many more. For example, for 8x8 blocks one possible distance function could be

Figure 1: Block Diagram for a Conditional Replinishment Coder

we present an overview of the experiments we conducted. In section V we discuss the variants of the codec we tried. This is followed by some interesting discussion and results.

2 A Description of NV

NV belongs to a class of codecs which use conditional replinishment to transmit motion video. Many of the other video conferencing systems like inperson [6] and experimental ones studied by [5] [10] [1] also belong to the same class. Of these only inperson and network video are software codecs. Here we describe the various stages in NV (figure 1).

2.1 Frame Grabbing

Some of the important characteristics for the frame grabber are the size of the frame, it's resolution and the frame rate. When working with general purpose workstations the bus bandwidth also becomes an important factor. Thus we measure the frame rate as the number of frames that can be transferred to memory over the system bus. The range of the above parameters varies dramatically for the various workstations. Fig 2 shows the architecture for the *Indy* .

The video capture board for the *Indy* is capable of being programmed to provide data in various formats and sizes. The video subsystem digitizes the analog video, processes the digital pixels as requested, and then puts the pixels on the GIO64 bus [9]. The subsystem provides pixel formatting and DMA hardware.

The format of the data as captured by NV was in the abekas format (U,Y,V,Y for two pixels) with the chrominance data decimated by two in the horizontal direction (i.e abekas 4:2:2 YUV format). The image size is variable from a 640 x 240 to a 640 x 480 sized image at 30 fps.

This is the only hardware dependent part of the codec. If one has to port a system to a lot of machines then it would be natural to structure the codec in such a way that the interface with the

The quality of video is affected by a variety of factors. The two broad cateogaries being computation and compression. It is important to study the compression and computation characteristics of such a system. By computation characteristics we mean the memory behaviour and the instruction costs. Most video conferencing systems stretch the power of a workstation to its limit, thus it becomes crucial to appropriately partition a computers scarce resources. The quality of video communication is very strongly dependent on the manner of this partition. In this paper we analyze NV and discuss how it utilizes the computers resources in particular for the *Indy* . From our analysis we suggest changes which result in a significant improvement in performance. Our analysis may be used as a guide to understanding the principal tradeoffs that a video conferencing application has to face. The results of our analysis are quite general and are pertinent to any software based video conferencing system.

Many vendors today supply video cameras and capture boards as part of the workstations. Of the systems available NetVideo (NV) from Xerox Parc is undoubtedly the most popular one [3]. It is popular and in wide use on the internet. It has been ported to a variety of platforms and has been used to multicast the conferences, like the IETF live over the MBONE. Thus we believe it is a representative example of a wide class of software based video conferencing systems.

We study the characteristics of each stage of the pipeline that captured frames go through before being transmitted over the network and how packets are decoded and displayed. The compression and computation characteristics of each stage are studied. The computation characteristics involve both the amount of instructions and the time spent due to cache misses. The breakup of the computation time into instruction and memory stall time shows us that the about 35 % of the time is spent waiting for the data to come into the cache on a system with a 16K primary cache and a 1MB secondary cache. We are able to reduce this to about 6-10 % of the total time by some modifications to the data structures.

We study the motion detection algorithm used by NV and come up with a scheme to evaluate the performance of any motion detection algorithm. It helps to find the optimum operating point, which we show can also be found by using the feedback due to degraded quality on the number of blocks sent. The algorithm used by NV is shown to be robust and close to optimal.

A software codec can either run out of bandwidth or out of processing power. We experiment with using different compression schemes for different range of bandwidths. For low bandwidths we use a DCT based scheme while for higher bandwidths we use the scheme used by NV which is less computationally intensive.

On most general pupose machines it is impractical to do processing on a bit stream. All variable length codes like the huffman code or arithmetic coding involve operating on bit streams and this is too inefficient. [1] Thus a surprising result comes up that the quantization step used in JPEG [11] like algorithms doesn't really help us in getting compression and we can get the same amount of compression by using simple thresholding and thus resulting in much better picture quality.

In order to guide the development of systems which support video conferencing, we study the effect of varying the cache line size and the size of the total cache on the system.

It is difficult on most workstations today to capture and compress video at 30 frames per second (*fps*). The characteristics of the encoded video might change if the stream is captured at 5-15 *fps* . We study the effect of decreased resolution in the temporal dimension on the compression algorithms.

The paper is organized as follows. In Section II we describe NV's algorithm and present a dataflow diagram explaining its operation. In Section III we present our evaluation methodology. In section IV

[1] We know of fast implementations of entropy coding, but since we are on the limit of the computation we can perform, these are costly in our scenario

Performance of Software Video Conferencing Systems: A Case Study of NV

Rosen Sharma
Center for Integrated Systems
Stanford University

June, 1994

Abstract

This work studies the performance of a software based video conferencing system. We study each stage from the capturing of the frame to its packetization and are able to isolate bottlenecks in performance. It was found that over 35% of the time was spent due to cache misses, waiting for the video data to appear in the cache. We are able to reduce this time to 6-10 % of the total time. For different bandwidths the bottleneck shifts between compression and computaton. We explore different compression algorithms for different bandwidths. We suggest an evaluation methodology for the motion detection algorithm used in any system and use it to determine a good operating point. We also study the effect of reduced temporal resolution in terms of frame rate on the compression algorithm, as most workstations today are not able to capture and process data at 30 frames per second. Entropy coding like the hufman code are costly to do, as they operate on bit streams. In situations like ours where we don't do entropy coding we show that simple thresholding gives same amount of compression as quantization but at a much better picture quality. Finally we study the effect of changing architectural parameters like the line sizes of the primary and secondary and the size of the primary cache on the performance of the system.

1 Introduction

Video conferencing on the desktop today is a reality. The multicast service on the Internet has lead to the development of video conferencing systems which are run on general purpose workstations. Video conferencing has been often dubbed as the "killer app" of the nineties. Motion video generates huge amounts of data, for instance, an uncompressed stream of a 24 bit (RGB), 320x240 video sequence at 30 frames per second requires a bandwidth of about 55 Mbits per second. Compression Algorithms, on the other hand, test the CPU's computational capacity. Thus most codecs would tradeoff computation with the compression.

Video Conferencing systems can either be software or hardware based. Hardware based systems offer better quality in situations where we are bound computationally by the processing power of the general processor machine. However the use of hardware based schemes limits the experimentation that can be done with them and also their scope of use. Software based schemes are flexible and are easier to port and thus enjoy a much wider audience. Also we believe in the philosphy that it is always better to use general purpose processors rather than specialized ones. The advantage being that CPU's are getting faster and also the general purpose hardware can be used by other applications. This doesn't mean that no special support is needed from the hardware, but rather we should be careful in what to place in the two domains.