A Programmable Image Compression System

Paul M. Farrelle

Optivision, Inc.
2655 Portage Bay Ave., Davis, CA

Abstract

This paper describes a programmable image compression system which has the necessary flexibility to address diverse imaging needs. It can compress and expand single frame video images (monochrome or color) as well as documents and graphics (black and white or color) for archival or transmission applications. Through software control the compression mode can be set for lossless or controlled quality coding; the image size and bit depth can be varied; and the image source and destination devices can be readily changed. Despite the large combination of image data types, image sources, and algorithms, the system provides a simple consistent interface to the programmer. This system (OPTIPAC™) is based on the TI TMS320C25 digital signal processing (DSP) chip and has been implemented as a co-processor board for an IBM PC-AT compatible computer. The underlying philosophy however can readily be applied to different hardware platforms; and by using multiple DSP chips or incorporating algorithm specific chips the compression and expansion times can be significantly reduced to meet performance requirements.

Introduction

The goal of image data compression is to squeeze out the redundancy in a digitized image (B bits/pixel) such that the compressed image can be represented by $b < B$ bits/pixel, but can later be expanded to give an output image containing B bits/pixel. The ratio B/b is called the compression ratio (CR) and the challenge is to maximize the CR given a set of constraints. Typical constraints are: lossless fully reversible compression so that the output and input images are bit for bit identical; a fixed maximum execution time; a minimum subjective output quality level.

Image data compression has become an integral part of imaging systems despite improvements in transmission and storage capacities. This is because imaging systems continue to use higher resolution sensors and the size of image data bases is growing rapidly. In some cases it is economically attractive to incorporate compression into an imaging system, while other times its use is mandatory to achieve the overall system specifications.

In this paper we describe a compression and expansion system which is rapid but does not operate at real-time rates. We have previously reported a compression system suitable for real-time TV and videoconferencing applications [1]. The current system processes each frame without reference to any other frame and is therefore an intraframe, or single frame, coder. Single frame imaging applications include: telemetry, remote surveillance, image databases, picture ID systems, medical

imaging, law enforcement, electronic publishing, digital mapping, insurance claims, parts catalogs, point of sale systems, etc. [2]. Each application has different requirements and even within a single application there is often the need to handle multiple image data types, sources, and compression modes. Image data types might include monochrome with 8, 10, or 12 bits per pixel; color with 16 or 24 bits per pixel; and binary images which can only take one of two values. Image sources are also diverse and images could come from computer files, frame grabbers which can capture a single video frame, image scanners, and communication links. Finally the application might require a lossless compression mode, or this constraint might be relaxed to allow small differences between the output and original images to increase the CR. In this controlled quality case there is a trade-off to be made between the output image quality and the CR. In either case the mode can be further specified as being high speed or high compression ratio. In the former case the algorithm is kept simple so as to achieve a high speed compression or expansion. In the second case the emphasis is to achieve the highest CR possible and as expected this will take longer.

It is clearly desirable to have a single system which can handle multiple image data types (binary, gray scale, color) at multiple resolutions and provide different compression modes (lossless, controlled quality). The overall system requirement is then to provide all of this flexibility via a consistent interface.

OPTIPAC™ Hardware Specifications

A system has been built to provide this capability in a PC environment. It is a co-processor board for an IBM PC/AT compatible computer and is built around the TMS 320C25 DSP chip operating at a 40MHz clock speed. Local memory consists of 32 or 64 kbytes of high speed static RAM with no wait states. The memory is used to store the current compression application and provide a data area where the current image window can be processed. The memory is dual ported so that it can be accessed by both the DSP and the AT host. The AT downloads the appropriate compression or expansion application code at the beginning of a session and then compresses an image, a window at a time, by repeatedly loading image windows and unloading compressed data blocks. For more exacting requirements, multiple OPTIPAC™s can be operated in parallel, within a single system, to reduce the overall execution time.

Software Requirements

The major requirement was to provide a consistent interface which would be independent of the compression algorithm, the image data type, and the image source and destination device types. A second important requirement was to allow algorithms to be selected via descriptive terms such as controlled quality, level 6 rather than "optimal adaptive widget coding with a threshold of 32.452 followed by customized variable length coding". This has the obvious advantage of not requiring a user to be a coding expert, but it also provides one level of indirection so that different algorithms can be substituted at a later time without changing high level application code.

Indirect Algorithm Selection Table (IAST)

To implement the descriptive approach to algorithm selection, we created an Indirect Algorithm Selection Table (IAST). This table translates descriptive terms plus the image data type into a 4-bit index which then maps into a specific algorithm. By combining this table with the DSP application code, algorithms can be changed or updated without recompiling any of the user's application code. All that is required is a new DSP application file which can readily be distributed to existing users. The current IAST is shown in Table 1 and the algorithms which have been implemented are characterized as follows [3]:

A10 - Lossless high compression ratio algorithm for monochrome or RGB color images. Based on predictive and variable length coding.

A20 - High speed lossless coding algorithm for monochrome or RGB color images. This is also based on predictive and variable length coding, but uses simpler versions to increase the speed.

A30 - High compression ratio controlled quality algorithm for monochrome images. Based on adaptive cosine transform coding.

A31 - Same as A30 but for RGB color images.

A40 - High speed lossless coding algorithm for binary images. Based on CCITT one-dimensional RL coding.

A50 - Lossless high compression ratio algorithm for binary images. Based on CCITT modified READ coding.

The Universal Algorithm Interface (UAI)

The OPTIPAC™ compression system is shown in Fig. 2. In this diagram the PC host memory is represented by the central block and contains a compression application which has been linked with the OPTIPAC™ run time library to form an executable image. The interface between the user's application and the run time library is depicted as the universal algorithm interface (UAI) and represents a set of function calls which are used to control a compression or expansion session. At the top left of the diagram we see image data stored in a file and also a frame buffer. The *display.cnf* file associated with the frame buffer is simply a configuration file needed by the system. At the top right we see the destination for the compressed data, a second disk file. This is simply one example configuration and different sources and destinations are of course possible. At the bottom of the diagram we see the compression hardware and a disk file which contains the DSP applications and the IAST.

Stage I - Setup The first stage of a compression session is the setup. This is accomplished by a call to the SetupCompress function using the following syntax:

SetupCompress(*x, y, z, nx, ny, nz, color, bits, type, mode, quality*)

This specifies to the compression system that we wish to process a region of size (*nx, ny, nz*) at offset (*x, y, z*) where *x* and *y* are spatial coordinates within the current frame, and *z* is the frame number. The image data type is specified by *color* which specifies color or monochrome and *bits* which is the number of bits per pixel. Then finally the *mode* and *quality* parameters specify the compression mode and one of the ten controlled quality coding levels: 9 (highest), 8, ..., 0 (lowest). This function call initializes the compression code and, as shown in Fig. 3, causes the appropriate DSP application (algorithm) to be downloaded to the compression engine.

Stage II - Compress The next step is to compress the image region specified in the setup stage. This is accomplished by a call to the Compress function using the following syntax:

Compress(*read_window, write_block*)

This provides the compression system with two user supplied I/O functions: *read_window* and *write_block* which are independent of the specific algorithm in use. In Fig. 4 we have:

Compress(*ReadFileWindow, WriteFileBlock*)

and *ReadFileWindow* is called by the run time library to load image data into the compression hardware and then, after processing, the compressed data is stored using the *WriteFileBlock* function. By simply changing the I/O functions, unlimited image sources and destinations can be accommodated. For example in Fig. 5 we have:

Compress(*ReadFrameWindow, WriteFileBlock*)

and the image source is now a frame buffer rather than a disk file.

Since the compression hardware is unable to store the complete image on board, the *read_window* and *write_block* functions are called repeatedly to process the complete image region specified in Setup. Furthermore the window size requested each time by OPTIPAC™ is algorithm dependent and is made as large as possible to minimize the overhead associated with a data transfer between the PC host and OPTIPAC™. The run time library code handles this complicated algorithm dependent control leaving a simple consistent interface, the UAI. The complete system is shown in Fig. 6.

Performance

Overall system performance is shown below in Tables 2 and 3. In Table 2, CR is shown when images with different data types and of differing complexity are compressed using some of the available modes. Five different compression modes are shown: **hs** - high speed lossless; **hc** - high compression ratio lossless; and q5, q3, q1 which are controlled quality coding at quality levels of 5, 3, and 1 respectively. The corresponding execution times are shown in Table 3.

Typical times for compressing 512x512 monochrome images on a standard 8 MHz IBM PC/AT are: 2 seconds for the **hs** mode and 6 to 7 seconds for the controlled quality mode. Coding 512x512 color images takes 5 to 6 seconds for the **hs** mode and 9 to 10 seconds for the controlled quality mode. Compressing binary 8.5"x11" pages sampled at 200 dots per inch and 200 lines per inch takes about 2 seconds for **hs** and 4 seconds for the **hc** mode. Expansion times are similar to compression times.

Note that although the CR figures for **hc** coding are always greater than, or at least equal to, the corresponding **hs** figures, they are often not significantly larger for monochrome and color images. In fact, **hc** only provides substantial improvements when the **hs** figures are already relatively large. Furthermore **hc** coding can take much longer than the other modes because it is currently implemented on the slow PC/AT host. If the application is long term archival and the only requirement is to obtain the highest possible CR then the **hc** option should always be used. Otherwise, only when the noise level is low and there is a great amount of redundancy in the image, that is the **hs** CR is 2 or more, is it usually worth spending the extra time to utilize the **hc** option. Consider the color baboon and logo images as extreme examples. The baboon takes an extra 72.5 seconds to compress in **hc** mode rather than **hs** mode, but still only achieves the same CR (=1.4). On the contrary, the computer generated logo image takes only an extra 10.5 seconds to dramatically increase the CR from 2.4 to 23.2.

Conclusions

We have described a programmable image compression system that can handle images of any type and size while maintaining a consistent interface. For systems requiring even faster performance, multiple boards can be used or the consistent design philosophy can be extended to more complex systems containing multiple processors and compression specific hardware modules.

References

1. Jain A. K., and D. G. Harrington, "A 10 MHz data compression system for real-time storage and transmission", Appl. of Digital Image Processing VIII, Proc. SPIE 575, pp. 62-65, 1985.

2. Farrelle, P. M., D. G. Harrington, and A. K. Jain, "Image data compression in a personal computer environment", Appl. of Digital Image Processing XI, Andrew G. Tescher, Ed., Proc. SPIE 974, pp. 177-186, Dec. 1988.

3. A. K. Jain, "Fundamentals of digital image processing",Chapter 11, Prentice Hall, 1989.

graphics	color	high compression	controlled quality	index	algorithm
0	0	0	0	0	A20
0	0	0	1	1	A30
0	0	1	0	2	A10
0	0	1	1	3	A30
0	1	0	0	4	A20
0	1	0	1	5	A31
0	1	1	0	6	A10
0	1	1	1	7	A31
1	0	0	0	8	A40
1	0	0	1	9	A30
1	0	1	0	10	A50
1	0	1	1	11	A30
1	1	0	0	12	A20
1	1	0	1	13	A31
1	1	1	0	14	A10
1	1	1	1	15	A31

Table 1 Indirect Algorithm Selection Table (IAST). The image data type (graphics, color) and compression mode (high compression, controlled quality) are used to form a 4-bit index into the IAST. The first bit is 1 for graphics images and 0 for non-graphics images; the second bit is 1 for color and 0 for monochrome; the third bit is 1 for high compression ratio mode and 0 for high speed mode; the fourth bit is 1 for controlled quality coding and 0 for lossless coding. The specific algorithms (A10, A20, A30, A31, A40, and A50) are described in the text.

	hs †	hc ‡	q5 *	q3 *	q1 *
Monochrome images (512x512x8)					
Very Simple Scene (adac)	2.8	3.9	32	45	115
Simple Scene (f18)	1.7	1.9	14	21	49
Complex Scene (airport)	1.3	1.5	8	12	31
Color images (512x512x16)					
Very Simple Scene (AT&T logo 512x400)	2.4	23.2	31	46	111
Simple Scene (lenna)	1.8	2.0	22	35	92
Complex Scene (baboon)	1.4	1.4	10	17	61
Black and White images (1728x2376x1)					
Simple Scene (CCITT 2)	12.5	18.8	-	-	-
Complex Scene (CCITT 7)	4.5	5.3	-	-	-

† hs - High Speed lossless compression mode
‡ hc - High Compression Ratio lossless compression mode
* q5, q3, q1 - Controlled Quality (Quality Levels: 5 (highest), 3, 1 (lowest))

Table 2 Compression ratios (CR) for different image data types and compression modes.

	hs †	hc ‡	q5 *	q3 *	q1 *
Monochrome images (512x512x8)					
Very Simple Scene (adac)	1.8	15.5	5.8	5.6	2.4
Simple Scene (f18)	2.2	29.8	7.6	6.7	3.0
Complex Scene (airport)	2.1	36.4	10.1	8.5	3.6
Color images (512x512x16)					
Very Simple Scene (AT&T logo 512x400)	4.5	15.0	7.5	7.1	3.5
Simple Scene (lenna)	5.9	63.5	10.4	9.0	3.6
Complex Scene (baboon)	5.9	82.4	15.5	12.0	4.1
Black and White images (1728x2376x1)					
Simple Scene (CCITT 2)	1.7	1.8	-	-	-
Complex Scene (CCITT 7)	3.9	4.6	-	-	-

† hs - High Speed lossless compression mode
‡ hc - High Compression Ratio lossless compression mode
* q5, q3, q1 - Controlled Quality (Quality Levels: 5 (highest), 3, 1 (lowest))

Table 3 Compression times (secs) corresponding to CR figures shown in Table 2. Note that all times are memory to memory on an 8 MHz IBM PC/AT.

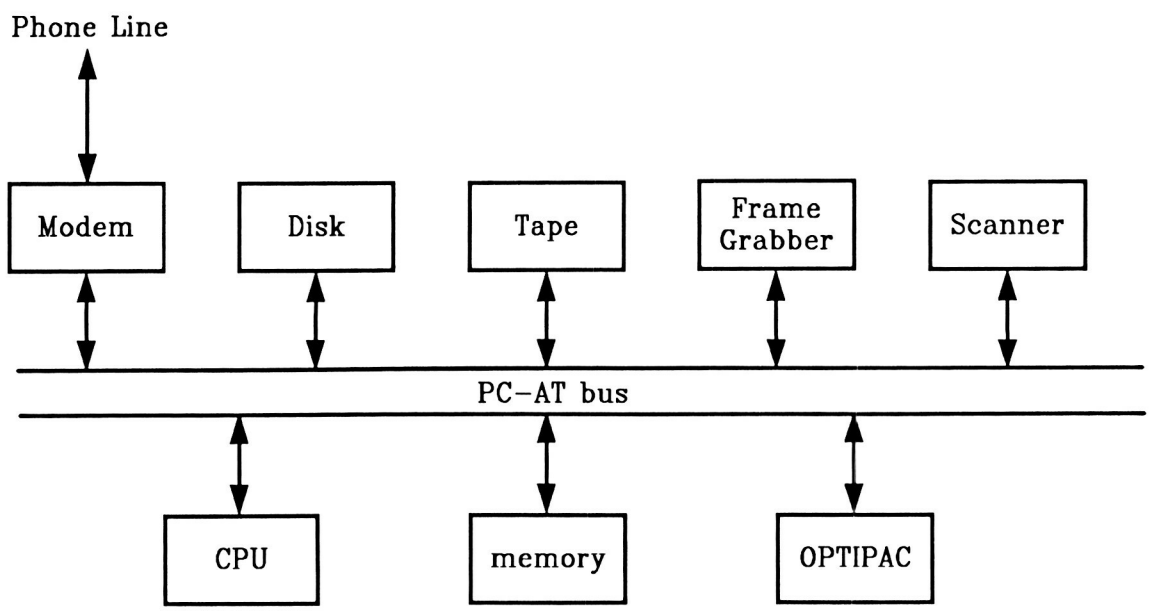

Figure 1 OPTIPAC™ in a PC environment. The input (or compressed) image from a disk, tape, image/document scanner, modem etc., is read into the main memory. It is then compressed (or expanded) by the OPTIPAC™ and routed to the output via main memory.

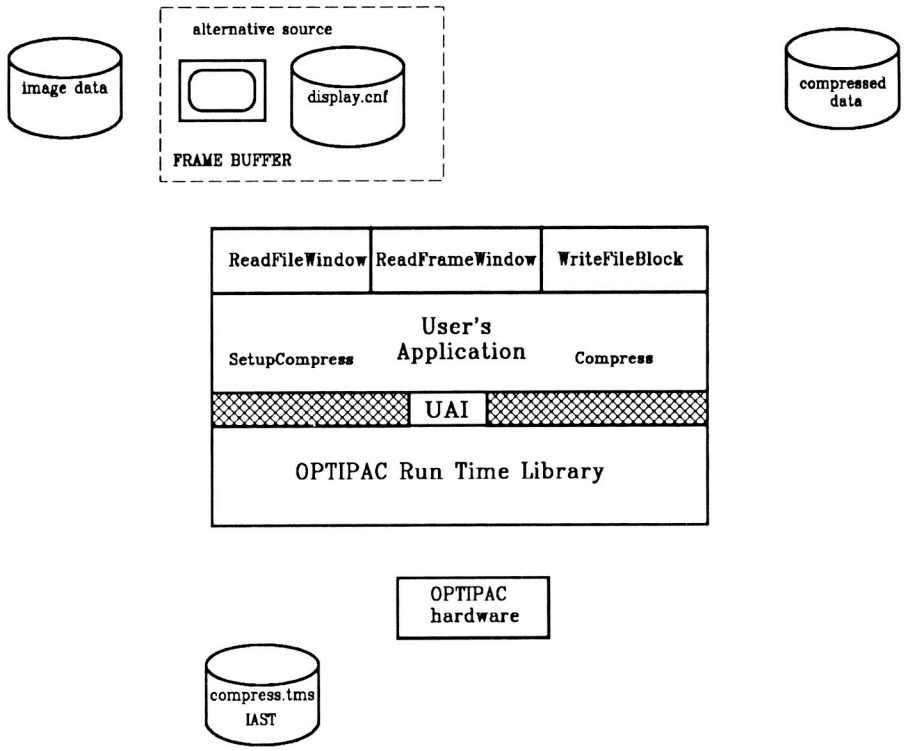

Figure 2 An overview of the OPTIPAC™ compression system.

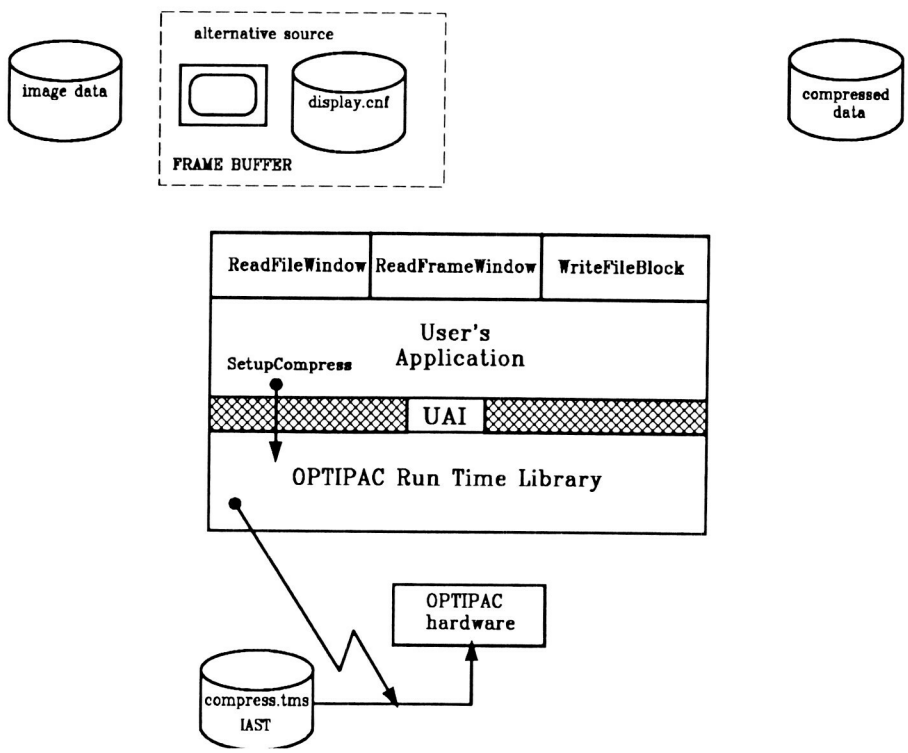

Figure 3 SetupCompress - selecting and loading the appropriate compression algorithm from *compress.tms*, the DSP application library. Note that this data file also contains the IAST which is therefore independent of the application code.

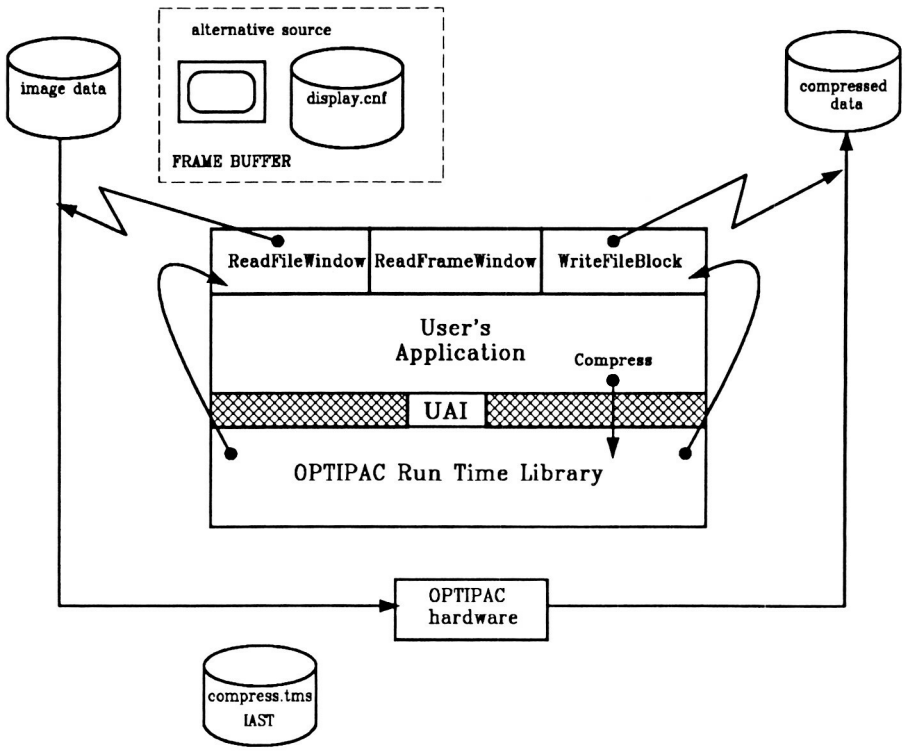

Figure 4 Compress - compressing an image stored in a disk file to a second disk file. Note how the run time library uses I/O functions provided by the user to access image data. In this way any I/O device is readily accommodated.

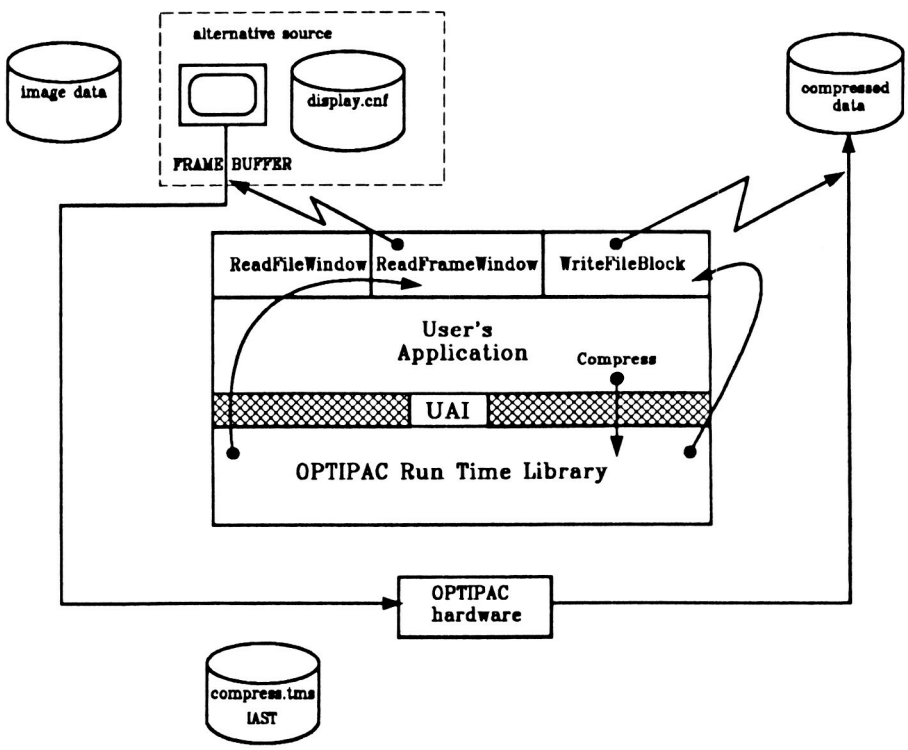

Figure 5 Compress - compressing an image stored in a frame buffer to a disk file. Note the similarity with Fig. 4, the only difference is that *ReadFileWindow* has been replaced by *ReadFrameWindow*.

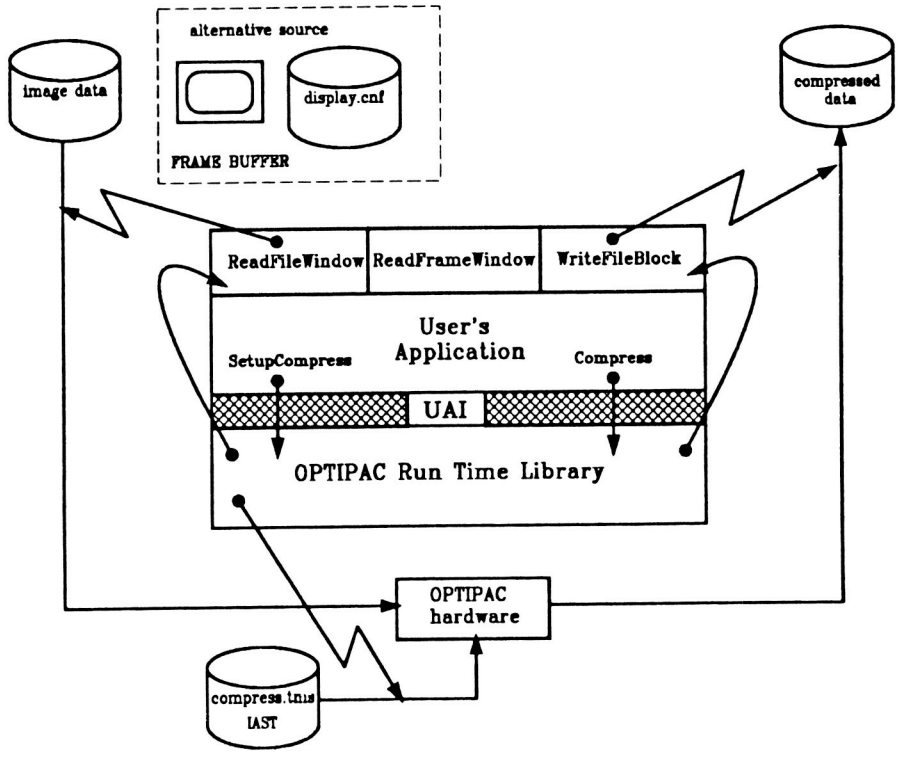

Figure 6 The complete OPTIPAC™ compression system showing the combined effects of the two stages: SetupCompress and Compress.