THE PERFORMANCE OF THE DIPS IMAGE PROCESSING SYSTEM

Othman Alhusain

Research Fellow, Technical University of Budapest, Dept. of Photogrammetry, Hungary, ISPRS commission II.

ABSTRACT:

In this paper, the performance of a digital image processing system called DIPS is measured in relation to different operating systems and hosting hardwares. The measurements were carried out under VAX/VMS, Unix, and MS-DOS operating systems and their suitable environments. A performance comparison between the results obtained proved, however, that the efficiency of the system depends not only on the hardware capabilities but also on the software resources whether they were operating systems or program code implementation so further performance measurements were done on the software through the analysis of the program code during the execution phase, the analysis proved that although the bulk of time is spent in execution the main loop of each routine, a considerable time is also spent within the assisting loops.

KEY WORDS: Evaluation, Performance, Systems, Measurements, Profiling.

1. INTRODUCTION

During the last decade computing hardware has witnessed rapid technological developments resulting in continual improvement in performance and reduction in cost of the microcomputer systems. The shift from mainframe and minicomputers to inexpensive powerful PC's for image processing in remote sensing was a natural consequence of the user's desire to have a dedicated, though powerful system for his personal use (Ferns, 1984; Myers and Bernstein 1985; Welch, 1989).

Image processing in remote sensing is constrained, however, by a number of processing requirements, including that remotely sensed images are multispectral, consisting of multi-band images with sizes much larger than standard CCD images and they are inherently "fuzzy"; their interpretation depends on a priori information, consequently image processing systems for remote sensing have to be efficient and capable of large data file handling (Ferns and Press, 1988; Muller, 1988; Ehlers, 1990).

Analysis of a system utilized for remotely sensed data processing is of great importance due to the nature of this image data, to the procedures applied to them during the manipulation process, and to the varying conditions dictated by any newer working environment surrounding the system. In order to cope with the previous factors and their impact on the ideal use of the processing system, an integrated analysis of the DIPS system was carried out, the analysis procedures were made through two types of measurements, those are hardware and software oriented performance measurements.

2. HARDWARE ORIENTED MEASUREMENTS

The performance of digital image processing system may be influenced by the system's I/O throughput for I/O intensive applications or by the efficiency of the code generated by the programming language for CPU-limited applications (Croswell and Clark, 1988; Freiman, 1988).

To enable comparison, parts of the DIPS code are generated to be implemented under VAX/VMS, UNIX and MS-DOS operating system.

2.1 Hardware Environement

VAX-related measurements were carried out on a MicroVax system with 4 Megabyte of memory and a floating point processor. Under VMS all image files were contigous files. The VMS system was supported by its Fortran compiler and Whitesmiths compiler.

PDP-11 measurements were carried out on a PDP-11/44 with 0.5 Megabyte memory and a floating point processor, the PDP was equipped with two disks type RL02, where the system's files reside on one of them while the user's files reside on the other. Under PDP-11 Unix the file systems were also interleaved as recommended for optimal use. The PDP-11 Unix system was standard V.7.0 using the Fortran compiler and the Ritchie C compiler.

Measurements on microcomputers were carried out on a system with an Intel 80286/16 bit microprocessor with an 80287 as a coprocessor. The operating system is MS-DOS and files are resident as binary values on the system hard disk.

2.2 Measurements

The procedure to measure the performance of the systems consists of the following two steps:

1. The I/O performance of each operating system is measured by reading a file using the most efficient I/O available for each operating system and each type of I/O.

2. The performance for a typical operational loop in the DIPS system is measured for each form of the I/O supported by the system and for different buffer sizes. The aim of the loop operation is to read image pixels and to convert them to floating point format. The measurement procedures are as follows:

MicroVax: In a maneuver to measure the optimum I/O of different programs written in different languages and working under VAX/VMS, an image of 512 × 512 pixels was read without processing which involves reading the command line by the system and executing the reading process. By using double buffered I/O the reading operation required from VMS 1 second. 2 seconds were required to execute the same operation executed by the following "read from file" function from the DIPS system:

```
nbytes = (int) sakf((double) (npix*(*out)--> nbit)/
       (double)TCRC;
  for (dukik = j = 0; j < nlin; j ++) {
   for (i = 0; i < nbnd; i ++) {
    dukik + = fread((*out)--> data[i][j], nbytes,
       1, fp);
 if ((*out)--> nbit < sizeof (PIXEL)*(TCRC) LARUZM
```

```
        (npix, (*out)--> nbit, (*out)--> data[i][j];
            }
    }
```

8 seconds were required to execute the same operation by Fortran 77 code using a direct-access file with a record size of 512 bytes and a block size of 8 Kbytes to improve efficiency. These Measurement results show that the I/O efficiency of the DIPS C implementation is high indeed; it is 4 times higher than the Fortran implementation, and half the optimal I/O efficiency of the VMS operating system.

To investigate both the buffer size and I/O type that accomplish the optimum use of the DIPS system under VMS operating system. Originally DIPS was written supporting block I/O only, for this reason the required code was generated to support both Unix I/O and virtual I/O. The operation was the same: reading an image or file of 512×512 pixels but this time under different buffer sizes through the three Unix, block, and virtual types of I/O. The results are listed in table 1.

It can be concluded from these measurements that the increase in buffer size results in a decrease in the I/O time and consequently higher performance of all types of I/O, however, under VMS the block I/O shows a slight advantage over Unix and virtual I/O. 8 and 16 Kbyte buffer sizes seem to be optimal for VMS block I/O.

Buffer size	Timing		
(KByte)	Unix I/O (sec)	Virtual I/O (sec)	Block I/O (sec)
1	7.0	4.2	4.0
2	5.0	4.0	2.0
4	5.0	3.7	1.8
8	3.0	3.5	1.6
16	3.0	3.0	1.6
32	3.0	2.8	1.5
64	3.0	2.8	1.5

Table 1, Buffer timing of the DIPS system

PDP-11: Unix implementation of DIPS supports block I/O and Unix I/O. During the measurement phase half of the RL02 disk was allocated to a block file system where 4 Kbyte space was assigned to each block, while the other half of the disk was allocated to a Unix file system. To determine the optimal I/O throughput of the DIPS system under Unix, a file consisting of 512×512 pixel bytes was read without processing by using the "read" system call and a buffer size of 8 Kbytes. The results of measurement are included in table 2; in these measurements, the CPU time is considered to be equal to the system time since the user time is equal to zero due to the fact that no processing was performed.

The next measurements were to verify the relationship between the buffer size and the Unix I/O of the DIPS system and so the optimum use of the DIPS system under the Unix operating system. The measurement operation goes through "reading" an image of 512×512 pixels without any processing. The results are shown in table 3. From the previous measurement results it is easy to notice that the CPU time decreases slightly while the buffer size increases, also the best clock time occurs when the buffer size is equal to the system block.

I/O type	System time (sec)	Clock time (sec)
Unix regular	3.0	9.0
DIPS-block	0.6	3.0
DIPS-Unix	0.6	3.0

Table 2, DIPS optimal I/O timing under Unix

Buffer size	Timing			
(KByte)	User (sec)	System (sec)	CPU (sec)	Clock (sec)
0.5	5.3	4.0	9.0	10.0
1	5.0	3.3	8.3	11.0
2	5.0	3.2	8.2	11.0
4	5.0	3.2	8.2	10.4
8	5.0	3.0	8.0	10.4
16	5.0	3.0	8.0	10.0

Table 3, DIPS timing with relationship to the buffer size
CPU time = User time + System time

Microcomputer: DIPS routines were originally developed taking into consideration that they will be run under Unix operating system. Due to two main reasons it was decided to implement many of the DIPS routines under the DOS operating system, the first reason is that DOS is still the most common operating system on the majority of microcomputer machines, the second reason is particularly that the microcomputer based on the Intel 80286 (or 80386) in our laboratories are not capable of managing such a large operating system as Unix together with the large files of remote sensed images.

DIPS routines under DOS operating system support only block structured files, as mentioned above, due to some restrictions of DOS itself, the image file consisting of 512×512 pixels was resident in the hard disk of the microcomputer as binary numbers, the buffer size was 32 Kbyte, the image was read without processing within 2 seconds which is optimum comparing to the DIPS Unix-11 implementation. The relationship between the buffer size and the time required to "read" the image file is illustrated by table 4. The measurement results in table 4 show that the increase of the microcomputer buffer size is associated with a drastic decrease of the "reading" time which is a very important advantage indicating the possibility of attaining a powerful I/O in the microcomputer system through the optimal selection of its buffer size.

Buffer Size (KByte)	CPU time (sec)	Clock time (sec)
0.5	6.0	6.3
1	5.0	5.2
2	4.5	4.7
4	4.0	4.2
8	3.0	3.2
16	2.0	2.1

Table 4, The DIPS/MS-DOS timing with relationship to the buffer size

3. SOFTWARE ORIENTED MEASUREMENTS

The aim of analysis measurements done on the software is to determine where the system spends its time within the different sections of the program during the execution stage and trying to avoid such loops as much as possible when designing the algorithms. A direct and easy method that enables such analysis measurements on the software can be done by the "routine calling occurrence" technique, that is to have the compiler generate instructions to count how many times each routine of the program has been called. This technique is easy to implement since it does not need any special requirement from the operating system, however it is not as accurate as possible since the number of times a routine is called is not necessarily indicating the period spent in executing that routine.
An alternative and more accurate approach to monitor the program's execution time is achieved by means of "profiling". A profiling technique involves mapping the addresses of specific sections of the program and counting the time of passing through these address through the execution course. This can be done simply by inserting statements that print a "start" and "stop" at the beginning and the end of a specific section and measuring the interval between the appearance of those "starts" and "stops". The profiling technique, in spite of its simplicity and effectiveness, is not provided under all environments and circumstances, however, the C programming language provides this feature through its conditional compilation utility which can be implemented under both Unix and DOS operating systems.

In order to carry out a real indicative analysis process on the DIPS system, and since it is agreed in the field of image processing that there is no typical operation which could be measured and analyzed in detail as a representation of the analysis of the system as a whole, 5 sample routines were selected from among the DIPS system routines, those routines are among those responsible for performing some of the most frequent operations in the field of image processing. The analysis measurements were obtained by implementing the profiling technique on a MicroVax system under VMS and on a microcomputer system under DOS and the results were similar.

The first of these routines is READIMG which represents the I/O operations carried out by the DIPS system, this routine reads an image from the storage of the system to the main memory. The execution of this routine proceeds through six steps; opening the image file, reading the image size parameters, creating an image of appropriate size and parameters, checking the parameters of the created file, reading the new file, and closing the image file. An image of 3 band, 512×512 pixels was read from file into memory, the execution process was profiled as mentioned above and gave the measurements listed in table 5.

As could be noticed easily, the time required to open the image file, reading the image attributes, and closing the image file again when completing the reading operation account for a very small fraction of time, while creating an image of appropriate size, checking the parameters of the created file and reading it account for the bulk of time (93%) required to execute the whole operation.

Stage Name	Timing Profile (%)
Open image file	6
Read parameters	3
Create image	44
Check parameters	18
Read new file	26
Close image file	3

Table 5, Timing profile of the READIMG routine

The second of the routines analyzed is ADDIMG which is one sample of similar routines called algebraic routines because they are responsible for carrying out algebraic operations such as adding, subtracting, and multiplying on images. This routine, although it is computationally simple, represents an actual and frequent image processing operation. The ADDIMG routine consists of three parts; these are checking the input images, creating an image of appropriate size, and adding the input images. The execution of the ADDIMG was measured through adding two images together each one is 3-bands and 512×512 pixels, the results are shown in table 6, where it is shown that the "adding" part of the routine consumes the maximum portion of the whole routine execution time (70%) and this portion is subject to increase if the operation is of a multiplication type. On the other hand the "image creation" part of the routine consumes a reasonable portion of time (21%), while the input image checking requires a minimum portion of time (8%).

Stage Name	Timing Profile (%)
Check input	8
Create image	21
Add images	70

Table 6, Timing profile of the ADDIMG routine

A fast Fourier transform operation was analyzed as an example of transformations that usually considered as complex algorithms but it is easy to implement them via software. These transforms used to be done frequently on images during the processing course and play an important role in the enhacement of image features. The analysis was done on the FFT routine on the same image size, the execution of the FFT routine proceeds through 9 stages,

those are check input image, creating an image of appropriate size, computing the nth roots of unity, allocating an array to hold one row/column of pixels, transformation of bands, converting from real to complex, transformation of rows, transformation of columns and finally, converting from complex to real. The execution profile of the FFT routine shows that the bulk of execution time is consumed during the transformation of bands, rows and columns with 19%, 26%, and 26% consequently converting from real to complex and from complex to real each one needs 7% of the execution time, 5% time is spent in computing the nth roots of unity and in allocating arrays.

Stage Name	Timing Profile (%)
Check input	2
Create image	2
Comput roots	5
Allocate Array	5
Transform bands	19
Convert real to complex	7
Transform rows	26
Transform collumns	26
Convert complex to real	7

Table 7, Timing profile of the FFT routine

The input image check and creating an image of appropriate size requires a minimum portion of time, about 2% for each. Table 7, summarizes these execution analysis results. It was noticed during carrying out the analysis process on the FFT routine that the "transform" parts of the routine consumes higher rates of time not only proportionally in comparison with other parts of the routine itself, but also absolutely if the total time needed to execute this routine is compared to the total execution time of other routines. The reason for this fact is that transform routines are done on a neighborhood of pixels instead of one, and so that when carrying any computation process a number of pixels are included together.

The next routine to be analyzed is the CONVL routine which is usually applied on images to improve their features via complex enhancement procedures especially by convolving them with filters. The convolution of an $(M \times N)$ pixsel image with a $(I \times J)$ element filter requires $(M \times N \times I \times J)$ multiplications. The example analyzed here was a convolution of a 1 band, 512×512 pixel image with a (3×3) element filter, this means that each output pixel is the result of multiplication on 9 input pixels. The CONVL routine consists of 6 execution stages, those are check input, create image of appropriate size, allocate box for filter, compute convolution, fill margins, and output result. The execution profile of the CONVL routine illustrated in table 8, shows that the maximum amount of time is spent in the "compute convolution" operation with 61%, fill margins requires 22%, allocate box for filter needs 8%, 4% of the time is needed for creating an image of appropriate size, check input and output result needed a minimum amount of time equal to 25% for each one. The analysis results including the proportional timing rates will be changed highly if the filter is changed e.q. applying a filter of (5×5) elements will result in a trend towards assigning higher timing rates for the "compute convolution" operation; this is due to the fact that when applying a (5×5) element filter the multiplications needed are 25 in comparison of only 9 multiplications needed in case of applying a (3×3) element filter.

The final routine of the DIPS system which was analyzed is the STATSIMG which computes the statistic data of an image such as the mean standard deviation, minimum, maximum, and range of gray level, and the values of the histogram. The STATSIMG routine implementation within the DIPS system consists of 4 parts, these are check input, compute band statistics, compute histograms, and output histograms. The execution profile of the STATSIMG, table 9, shows that "check input image" consumes a minimum portion of execution time of 4%, while the rest of execution time is divided almost equally among the remaining three parts of the routine with 36% spent within "compute band statistics" 31% within the "compute histograms" part, while the execution of the "output histograms" needs 27.5% of the whole execution time.

Stage Name	Timing Profile (%)
Check input	2
Create image	4
Allocate box	8
Compute convolution	61
Fill margins	22
Output results	2

Table 8, Timing profile of the CONVL routine

Stage Name	Timing Profile (%)
Check input	4
Compute band statistics	36
Comput histogram	32
Output histogram	28

Table 9, Timing profile of the STATSIMG routine

The analysis of the previous five examples shows by numbers that the amount of time required to execute each operation is divided among the different cycles and loops comprising the routine responsible for execution of that operation. However, although the bulk of the time is dedicated to execute the main loop of the routine, a considerable amount of time is also needed to execute the assisting secondary loops, so any effort towards attaining an efficient routine structure and consequently an efficient program should concentrate not only on improving the main loop but also on the supporting loops of that specific routine.

4. CONCLUSIONS

Digital image processing applications are described by large amount of data and intensive I/O operations, the whole performance of such systems may be determined by the I/O throughput or efficiency of the program code. Theoretically, it is supposed that the overall speed of a system increases proportionally with the number of its processors, however, measurement results presented in this paper showed that there is not a one-to-one gain in efficiency because the conflicting distribution of tasks among the multi processor of the mainframe. On the other hand performance attained by a one-processor microcomputer system proved to be encouraging and this fact will be more feasible if the fulfillment of the operating system in the microcomputer is utilizing the optimal abilities of its microprocessor. Profiling and analysis of different routines of the DIPS system show that the goal of attaining an efficient program could be realized only when the improving efforts are cocentrated on the main and assisting loops of routines.

REFERENCES

Croswell, P. L., and Clark, S. R., 1988. Trends in automated mapping and geographic information system hardware. Photogrammetric Engineering and Remote Sensing, 54(11): 1571-1576.

Ehlers, M., 1990. Low cost image processing on personal computers: the Macintosh-II based DIRIGO system. In:Int. Arch. Photogrmm. Remote Sensing., Dresden-GDR, Vol.28 Part, 2 pp. 55-62.

Ferns, D. C., 1984. Microcomputer systems for satellite image processing. Earth-orient. Applic. Space Technol., 4(4): 247-254.

Ferns, D. C., and Press, N. P., 1988. Microcomputer and mass storage devices for image processing. In: Muller, J-P., (Ed.) Digital Image Processing in Remote Sensing, Taylor and Francis, London and Philadelphia, pp. 105-122.

Freieman, S., 1988. A new dawn for workstations. Mini-Micro Systems, May, pp. 59-72.

Muller, J-P., 1988. Computing isues in digital image processing in remote sensing. In: Muller, J-P., (Ed.), Digital Image Processing in Remote Sensing, Taylor and Francis, London and Philadelphia, pp. 1-20.

Myers, H. J., and Bernstein, R., 1985. Image processing on the IBM personal computer. In: Proceedings of the IEEE, Vol. 73, No. 6, pp. 1064-1070.

Welch, R., 1989. Desktop mapping with personal computers. Photogrammetric Engineering and Remote Sensing, 55(11):1651-1662.