Performance Engineering of Image Processing Systems Through Benchmarking Techniques

Daniel F. García
Department of Informatics
University of Oviedo
Gijon, Spain
dfgarcia@uniovi.es

Francsico J. de la Calle
Department of Informatics
University of Oviedo
Gijon, Spain
UO224689@uniovi.es

***Abstract*—Currently, a multitude of systems are developed that process a continuous flow of images. They are used, for example, for video surveillance, inspection and control of products manufactured in production lines, automatic guidance of robots, vehicles, etc. For these systems to be useful, they must be able to process and store images at a minimum frequency. Therefore, when designing these systems it will be essential to determine the frequency at which they can operate depending on the characteristics of the images, the algorithms used to process them, and the computer hardware selected. Despite the importance of estimating and adjusting the performance of these imaging systems, there are hardly any methodologies for developing the performance engineering of them. With the aim of covering this gap, this article presents a simple experimental method to develop the performance engineering of the image processing systems and in particular to determine their maximum operational frequency or throughput. Using the proposed method, any performance engineer can determine the maximum working frequency of a system systematically and check that it exceeds the minimum required frequency. In addition, the engineer can also obtain the necessary information to reconfigure the system, eliminate performance bottlenecks, and take advantage of the computing power of the hardware properly.**

Keywords—image processing; performance engineering; benchmarking techniques

I. INTRODUCTION

Currently, the number of image processing applications is increasing more and more, for multiple purposes like continuous video-vigilance, entertainment, etc.

One particular sector is manufacturing, where image processing techniques are being extensively used to inspect and check the quality of manufactured products. Generally, the requirements for the development of new automatic inspection systems include a high defect-detection ratio, a low cost of the system, easy procedures for maintenance, etc. To meet these contradictory requirements, the system developers generally use standard image sensors and a powerful personal computer running a common operating system, such as Windows.

The utilization of standard computing hardware and operating systems allows lowering the cost of the system and makes maintenance easier, but it complicates the fulfillment of performance requirements, like the throughput or deadlines for processing the stream of acquired images.

The solution presented in this work is the utilization of a benchmarking technique. The developer must configure an adaptable benchmark to emulate the expected operational conditions of the target image processing application and its underlying hardware components. The experiments developed with the benchmark will provide insights about the maximum achievable performance.

Generally, the image processing system must achieve predefined minimum requirements to be useful, but if the system has sufficient computational resources, they can be used to improve the results provided. For example, the developer can weigh the achievable acquisition and processing rate of images against the size or resolution of the images used by the system.

Therefore, the developer can effectively use the benchmark to achieve an optimal compromise between the different configuration options for the system.

The rest of the paper is organized as follows: in Section two a review of related work is presented. The proposed benchmark is introduced in Section three and the performance engineering method is explained in Section four. Finally, Section five presents the conclusions of this work.

II. RELATED WORK

The estimation of achievable performance of image processing systems is a research topic of permanent interest. In this work, the term performance must be associated with the minimum execution time or the maximum throughput that a system can provide by optimally tuning the image processing algorithms to the underlying computing hardware. However, there are many other research works that use the term performance to describe the level of success of segmentation, classification, object detection, etc., of image processing algorithms.

The performance engineering process can be developed using techniques based on measurement, analytical modeling and simulation [1]. In general, an initial prototype of the system is required to take measurements of the execution time of the basic algorithms used in the system. Then, the

measurements can be used to adjust analytical or simulation models of the system.

Before starting a detailed software performance engineering process, benchmarking experiments can be developed to obtain insights about the achievable performance. But this performance depends on the computing architecture used to implement the system. There are several research works that study the benchmarking of image processing systems based on standard multi-core processors [2] [3] [4] [5]. Other works have extended the analysis comparing a classical superscalar processor with specialized architectures like SIMD and VLIW [6]. Furthermore, other works address the performance evaluation of systems developed on GPUs [7] [8].

All the previous works provide insights into how to evaluate the capabilities of general-purpose or specialized processors to execute image processing algorithms. However, the approach followed in this work is more particular, because the focus is on determining the achievable performance of an image processing application on a general-purpose multi-core processor using multithreading.

The benefits and limitations of parallel image processing techniques are summarized in [9], including many references to other research works. Any multithreaded application can take advantage of a multi-core processor in two basic ways: reducing the processing time of each image or increasing the number of images processed per second [10].

Several works have evaluated the reduction of the processing time of a complete image by partitioning the image in regions and devoting one thread to process each region, working all threads in parallel [11]. The partition can be defined manually and the threads are scheduled statically [12] or dynamically [13] to compensate the irregularities in the execution times provoked by slow threads. The partitions of the image can also be defined automatically using an API integrated with a tool like MATLAB [14].

There are very few works on methodologies focused on increasing throughput in image processing systems systematically. General ideas are presented in [15] and other works simply present high-throughput imaging systems [16] [17] but they do not explain their configuration or their computational optimization. Therefore, this paper covers a gap in the technical literature presenting a methodology for carrying out the performance engineering of image processing systems focusing on maximizing the achieved throughput.

Finally, with the measurements obtained from the prototype, a performance model of the image processing system can be developed and used for predicting the performance of the system [18], although this approach is complementary to that proposed in this work.

III. The Configurable Benchmark

The configurable benchmark is composed of three programs executed in two computers. The first program emulates the device that generates the images, a camera, and it runs alone in a computer. The second program emulates the driver of the device. Its primary objective is to receive images from the camera and place them in a shared memory. The third program is a consumer of the images placed in the shared memory. This consumer processes each received image using one or more algorithms sequentially, and optionally, stores the image on a disk. Both programs, the driver and the consumer, run on the computer initially selected to host the final image processing application. Fig. 1 shows this benchmarking environment, in which the two computers are communicated through 1 Gbps Ethernet link.

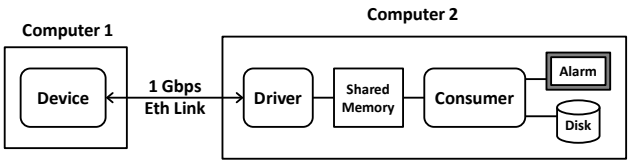

Fig. 1. Benchmarking environment

The following subsections explain the main components of the benchmarking environment.

A. The device emulator program

The program that emulates the camera operates as a network server that waits for a TCP connection from a driver. Just after the connection is established, the emulator starts a loop. In each cycle of the loop an image is generated and sent to the driver. Although a real device would be sending images until its disconnection or powering off, in this program a fixed large number of cycles is commonly used to develop the experiments.

At the beginning of each cycle, the program waits for the signaling of an event called StartCycle. This event is periodically signaled by a timer. The period between two successive signals can be easily configured.

Within the cycle, an image is generated with the required pixel format (graylevel8, rgb24, rgba32, etc.) and the desired number of rows and columns. The content of the image depends on the processing to be developed by the consumer. The images can have several backgrounds, such as random noise, patterns, gradients, etc., and they can have some foreground objects, like simple geometrical figures and text or numbers.

The image is then sent to the driver through the Ethernet link, using the sockets API and controlling the possible fragmentation of information. A number identifying the image is sent with each image.

Immediately after sending the image, the program waits for the reception of an acknowledge message from the driver, containing the identifier of the image received by the driver. Logically, the identifiers of the sent image and the received image must be identical.

Before the end of the cycle, the image generated is stored on a disk in a common format, like .bmp, to check the correctness of the generated images.

In order to check the performance of the device emulator, the time at the beginning and at the end of each cycle is stored

in arrays. At the end of the programmed cycles, the times required to generate and send the images are processed to obtain performance metrics.

This device emulator can also be used to develop a tolerance strategy against disconnection and reconnection of the Ethernet cable, network failure, etc., and also to emulate a control protocol for the device. However, these issues are not related to performance aspects, and therefore, will not be presented in this work.

B. *The driver program*

The program working as a driver of the camera operates as a network client that establishes a TCP connection with the camera emulator.

As soon as the connection is established, the driver starts a loop. At the beginning of each cycle, the driver waits blocked in a socket for the reception of an image. When the driver returns from the Receive call, it checks that all expected bytes have arrived, and then, it sends an acknowledge message to the camera emulator indicating the correct reception of the image. Afterwards, the driver writes the image in a shared memory, from which any consumer program can read it.

C. *The consumer program*

The consumer program follows a master-worker approach implemented with multiple threads. This software architecture allows the maximization of the system throughput easily. The main thread of the consumer waits, in a blocked state, for the driver to write a new image in the shared memory. When the driver finishes the writing of an image, it unblocks the consumer. Then, the consumer copies the image to a private buffer and starts a new thread to process the image. Optionally, a second thread can be started to store the image on a disk. Fig. 2 shows the expected behavior of this software architecture.

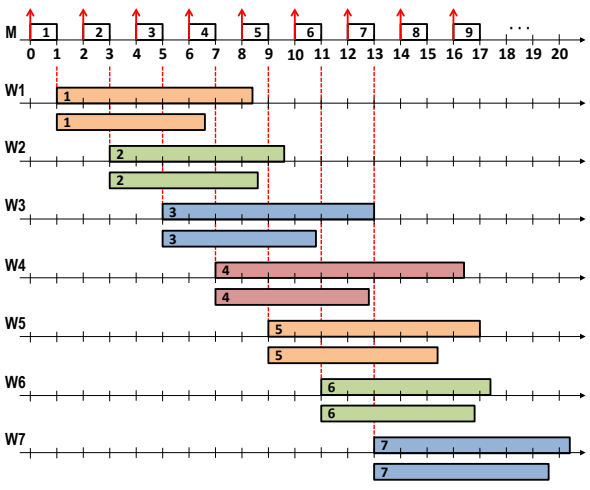

Fig. 2. Behavior of the master-worker application

In the prototype the processing algorithm must emulate realistically the processing required by the real application. The time taken by the processing depends on the computations carried out by the processing thread.

IV. The Performance Engineering Method

The primary objective of the method is to determine the maximum useful frequency achievable for an image processing system, implemented on a multi-core computer. But there is a broad objective beyond obtaining this frequency. The method must also provide metrics about the correct utilization of computer resources, giving indications about the presence of performance bottlenecks in software and hardware and the way to solve them.

In order to obtain the values of the metrics, a sequence of experiments must be developed. But before defining any sequence, each individual experiment must be defined.

A. *Characterization of a basic experiment*

Any experiment developed with the configurable benchmark can be modeled as the black-box shown in Fig. 3 with several inputs and outputs.

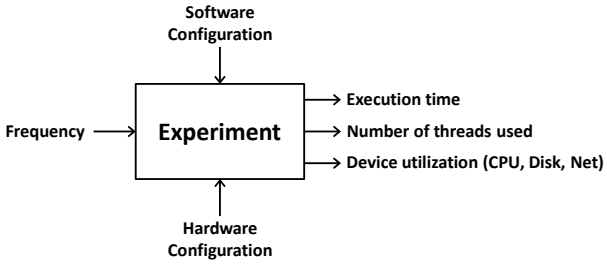

Fig. 3. Inputs and outputs of an experiment

Following the terminology used in Experimental Design, each experiment has response variables (outputs) and factors (inputs), whose influence on outputs must be quantified.

In Fig. 3 there are three response variables and multiple factors. At the top of the Experiment are the factors of the software, defined by the configuration of the image processing benchmark (image size, pixel format, image processing algorithms, etc.). At the bottom of the Experiment are the factors of the hardware, defined by the configuration of the computer used (processor type, number of cores, type and number of disks, etc.). On the left of the Experiment is the primary factor, the frequency of image generation, which defines the computational load supported by the system. This factor takes several values during an experiment, while the other factors remain constant.

B. *Determining the maximum operational frequency*

The method to determine the maximum operational frequency starts selecting the hardware and software configurations.

The main factors of hardware configuration are defined by the characteristics of the multi-core computer used to execute the software, which has an Intel Core i7 920 CPU at 2.66 GHz and 6 GB of DDR3-1066 RAM. There are two main factors of the software configuration. The first is related to the characteristics of the images, mainly defined by the image size and pixel format. For the basic experiments the size is 512 KB

using pixels of one byte (graylevel8). The second is the algorithm(s) to process each image, in this case, a median filter with a radius of two pixels applied to each image for smoothing.

With these factors selected, each experiment is developed varying the frequency of image generation. Typically, the engineer begins with a frequency of one image/second, and then progressively increases to two, four, etc., images/second exploring the whole range of frequencies at which the system can work in acceptable conditions.

The number of images to process in each experiment must be enough for the system to work on a stationary regime for sufficient time so that average values of output metrics can be calculated with a low variance.

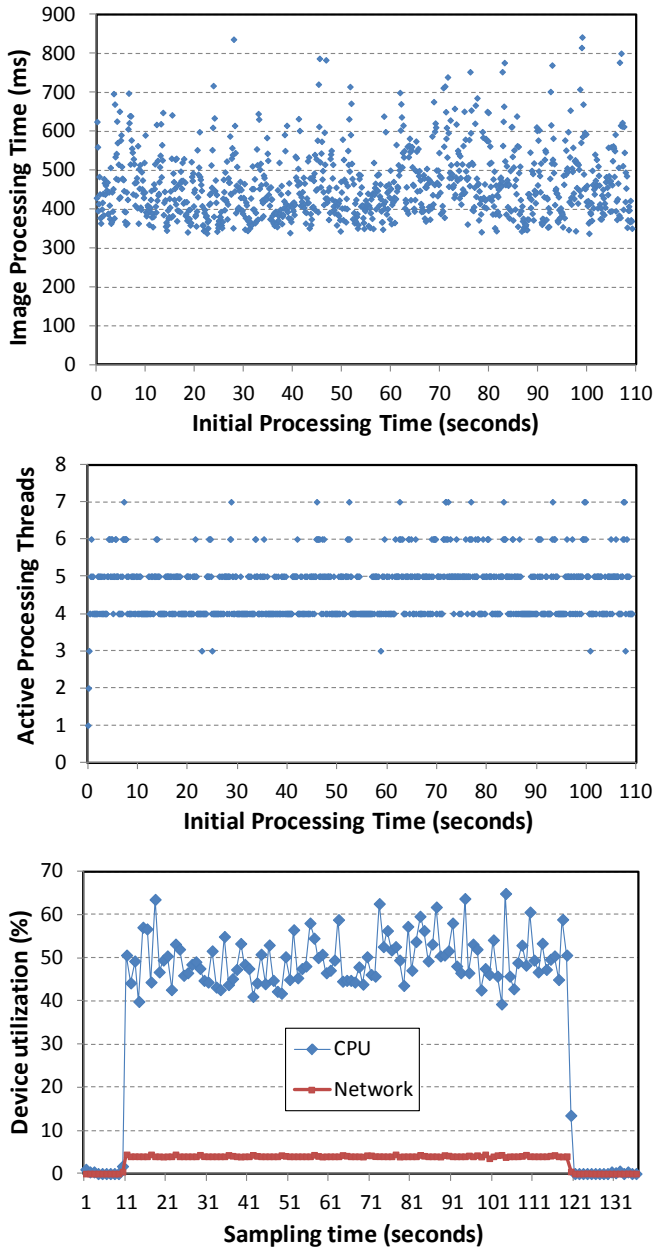

Fig. 4. Evolution of the main output metrics along an experiment

Fig. 4 shows the typical values measured along the time in any experiment: the processing time of each image, the number of threads active when the processing of each image starts, and the utilization of hardware devices.

From each experiment, the engineer must obtain the average values of the defined output metrics. With these averages, the graphs of Fig. 5 must be drawn by the engineer. All the graphs have the same horizontal axis, which represents the frequency of image generation, and all points of the graphs in a vertical line correspond to the same experiment.

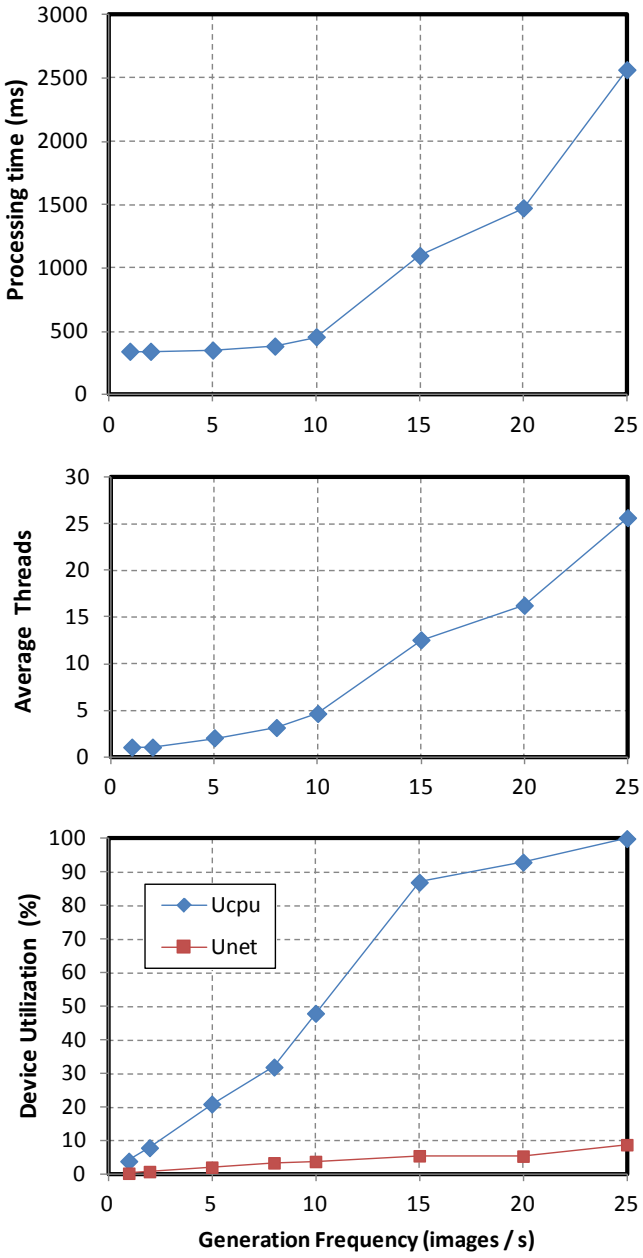

Fig. 5. Output metrics as a function of the generation frequency

The proposed methodology relies on the direct analysis of the graphs. The main output variable is the processing time, whose behavior can be related to the other output variables.

The average number of concurrent threads used to process the stream of images indicates the number of software resources dynamically deployed to process the stream. The utilization of hardware resources, mainly the CPU utilization, indicates how the software threads consume the available CPU time.

Three sections can be seen in the graph of processing time, which must be interpreted by the engineer in combination with the other graphs as explained in the following paragraphs.

In the left part, with a low frequency of image generation (<10 images/s), the system is underloaded, the processing time is the minimum possible, and increments of the frequency do not provoke perceptible increments of the processing time. The number of average active threads is low (<5) and does not reach the available processing units (8). The utilization of CPU is under 50%.

In the right part, with a high frequency of image generation (>10 images/s), the system is progressively overloaded and the processing time increases significantly when the frequency also increases. The number of threads running concurrently in the system is larger and increases because each thread needs a longer time to complete its task, due to the fact it has to share the processors of CPU with other threads. With high loads there is not one processor available per thread. The utilization of CPU increases quickly and soon reaches saturation (>90%) for a frequency of 15 images/s.

Between these two operational regimes, there is the inflexion point of performance, which can be considered the point in which the system changes from one regime to another (from low to high load operation).

Using the graphs, the engineer can select the frequency corresponding to the inflexion point as the maximum frequency achievable for the system. But this is the maximum frequency to guarantee the optimal performance, corresponding to the minimum processing time. In Fig. 5 this frequency is 10 images/second. However, the engineer could also select the lower frequency at which the CPU reaches saturation. From a practical point of view, the CPU is saturated when its utilization exceeds 90%. In Fig. 5 this frequency is approximately 15 images/second.

The performance engineer can select as the maximum operational frequency for this system any frequency between 10 and 15 images/second, depending on the main objective of the system.

Selecting a frequency close to 10 images/second, the processing times are the lowest possible, very predictable and with a low dispersion. However, only half of the capacity of the CPU is used.

Selecting a frequency close to 15 images/second, the processing times will be greater, very unpredictable and with a high dispersion. But with this selection the full capacity of the CPU is used.

C. *Sensitivity Analysis*

After finding the maximum operational frequency and knowing the behavior of the system as a function of the frequency, the engineer needs to know the sensitivity of the output metrics against variations in the input factors (software and hardware configuration).

This analysis can be done for the operational frequency selected for the system, for example 10 images/second. But it can also be developed for other frequencies building new curves. As an example, two software configuration factors are varied:

1) The size of the image, which is decremented and incremented by 25%, using image sizes of 384 KB and 640 KB instead of the original size of 512 KB.

2) A parameter of the image processing algorithm, the radius of the median filter. The initial value of this parameter is 2. In the sensitivity analysis this parameter is decreased to 1 and increased to 3.

Fig. 6 shows the processing times for the new values of image size. Clearly, the system is very sensitive to variations of the image size. Processing images of 640 KB, the CPU reaches saturation at a frequency of 10 images/second and with images of 384 KB, saturation is reached at a frequency of 20 images/second. Figures of the other metrics have not been included due to lack of space.

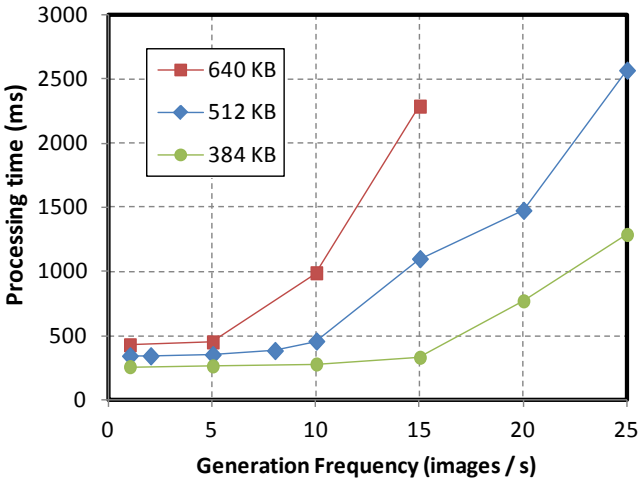

Fig. 6. Sensitivity of the processing time to variations of the image size

Fig. 7 shows the processing times for new values of the radius of the median filter. When the radius increases from 2 to 3, the CPU reaches saturation at a very low frequency, between 5 and 10 images/second, and the processing time increases to unacceptable values. On the contrary, when the radius decreases from 2 to 1, the CPU never saturates and the processing times practically do not change within the range of frequencies considered in this analysis.

Of course, this sensitivity analysis can be carried out for each of the software and hardware factors that the engineer considers could be changed. Furthermore, the engineer can analyze the effect of changing simultaneously 2, 3, or more factors. For a fixed frequency, the analysis of the influence of K factors which can take only two values (low, high) on a single output metric can be developed using the well-known theory of 2^K experimental designs.

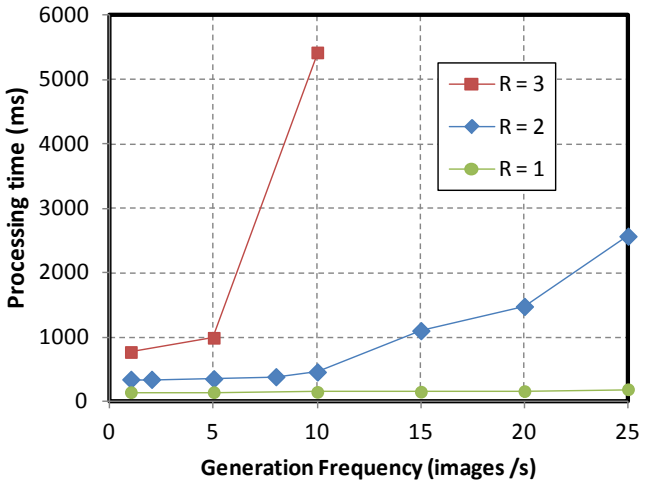

Fig. 7. Sensitivity of the processing time to variations of the radius of the median filter (the image processing algorithm)

V. Conclusions

In this work we have presented a method to develop the performance engineering of image processing systems. The proposed method is empirical and requires the development of an initial prototype of the application and the development of a sequence of experiments taking performance measurements. The method focuses on the determination of the maximum operational frequency of image processing, and optionally its storage, when the application is executed on multi-core computers.

The experimental environment required by the method is minimal, because it consists of only two computers. A multi-core computer is used to execute the image processing application. This computer must be similar to the final computer selected for the system. Another computer is required to execute the camera emulator. It is essential that this computer has enough computing power to generate images at the frequencies required by the experiments, without generating undesired delays.

Although the method may seem a little cumbersome, it provides very realistic performance metrics, and furthermore, the use of prototypes is always necessary to provide data to other analytic or simulation-based methods that could be less awkward.

Acknowledgment

This work has been partially funded by the project TIN2014-56047-P of the National Spanish Research, Development & Innovation Program.

References

[1] J. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley, 1991.

[2] J.W.V. Miller, C. Eddy, F.M. Waltz, R Hack, J. Wood and D. Stokes, "Image processing benchmark study," Proc. SPIE 3521, Machine Vision Systems for Inspection and Metrology VII, November 1998.

[3] P. Ranganathan, S. Adve and N.P. Jouppiy, "Performance of image and video processing with general-purpose processors and media ISA extensions," Proceedings of the 26th International Symposium on Computer Architecture, Atlanta, USA, May 1999.

[4] R.E. Artz, B.J. Loe, J.M. Pavelich and J.P. Bergmann, "Exploring multi-core processors using realistic signal- and image-processing application benchmarks," Proceedings of the 11th Annual Workshop on High Performance Embedded Computing, Lexington, Massachusetts, USA, September 2007.

[5] P.A. La Fratta, "Performance evaluation of synthetic benchmarks and image processing (IP) kernels on Intel and PowerPC processors," Technical Report RDMR-WD-13-11o US Army Research, Development, and Engineering Command, August 2013.

[6] D. Talla, L.K. John, V. Lapinskii and B.L. Evans, "Evaluating signal processing and multimedia applications on SIMD, VLIW and superscalar architectures," Proceedings of the International Conference on Computer Design, pp. 163-172, Austin, TX, USA, September 2000.

[7] I.K. Park, N. Singhal, M.H. Lee, S. Cho and C.W. Kim, "Design and performance evaluation of image processing algorithms on GPUs," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 1, pp. 91-104, January 2011.

[8] A. Asaduzzaman, A. Martinez and A. Sepehri, "A time-efficient image processing algorithm for multicore/manycore parallel computing," Proceedings of the IEEE Southeast Conference, Fort Lauderdale, FL, USA, April 2015.

[9] S. Saxena, S. Sharma and N. Sharma, "Parallel image processing techniques, benefits and limitations," Research Journal of Applied Sciences, Engineering and Technology, vol. 12, no. 2, pp. 223-238, 2016.

[10] S. Singh, P. Kaur, K. Kaur, "Parallel computing in digital image processing," International Journal of Advanced Research in Computer and Communication Engineering, vol. 4, no. 1, pp. 183-186, January 2015.

[11] C. Nicolescu and P. Jonker, "A data and task parallel image processing environment," Parallel Computing, vol. 28, no. 7-8, pp. 945-965, August 2002.

[12] A. Kika and S. Greca, "Multithreading image processing in single-core and multi-core CPU using Java," International Journal of Advanced Computer Science and Applications, vol. 4, no. 9, pp. 165-169, 2013.

[13] D. Akgün, "Performance evaluations for parallel image filter on multi-core computer using Java threads," International Journal of Computer Applications, vol. 74, no. 11, pp. 13-19, July 2013.

[14] Q.W. Samyan, W. Sahar, W. Talha, M. Aslam, and A.M. Martinez-Enriquez, "Real-time digital image processing using point operations in multithreaded systems," Proceedings of 14th Mexican International Conference on Artificial Intelligence, pp. 25-31, Cuernavaca, Mexico, October 2015.

[15] R. Petryniak, "Analysis of efficiency of parallel computing in image processing task," Czasopismo Techniczne Mechanika, vol. 105, no. 3-M, pp. 185-193, 2008.

[16] P. Kankaanpää, L. Paavolainen, S. Tiitta, M. Karjalainen, J. Päivärinne, J. Nieminen, V. Marjomäki, J. Heino and D.J White, "BioImageXD: An open, general-purpose and high-throughput image-processing platform," Nature Methods, vol. 9, no. 7, pp. 683-689, July 2012.

[17] A.C. Knecht, M.T. Campbell, A. Caprez, D.R. Swanson and H. Walia, "Image Harvest: an open-source platform for high-throughput plant image processing and analysis," Journal of Experimental Botany, vol. 67, no. 11, pp. 3587-3599, 2016.

[18] Z. Juhasz, "An analytical method for predicting the performance of parallel image processing operations," The Journal of Supercomputing, vol. 12, no. 1, pp. 157-174, January 1998.