DEVELOPMENT AND PERFORMANCE EVALUATION OF A LAN-BASED EDGE-DETECTION TOOL

Ghassan F. Issa[1], Hussein Al-Bahadili[1], and Shakir M. Hussain[1]

[1]Petra University, Faculty of Information Technology
P.O. Box 961343, Amman 11196, Jordan

(gisaa@uop.edu.jo, hbahadili@uop.edu.jo, shussain@uop.edu.jo)

ABSTRACT

This paper presents a description and performance evaluation of an efficient and reliable edge-detection tool that utilize the growing computational power of local area networks (LANs). It is therefore referred to as LAN-based edge detection (LANED) tool. The processor-farm methodology is used in porting the sequential edge-detection calculations to run efficiently on the LAN. In this methodology, each computer on the LAN executes the same program independently from other computers, each operating on different part of the total data. It requires no data communication other than that involves in forwarding input data/results between the LAN computers. LANED uses the Java parallel virtual machine (JPVM) data communication library to exchange data between computers. For equivalent calculations, the computation times on a single computer and a LAN of various number of computers, are estimated, and the resulting speedup and parallelization efficiency, are computed. The estimated results demonstrated that parallelization efficiencies achieved vary between 87% to 60% when the number of computers on the LAN varies between 2 to 5 computers connected through 10/100 Mbps Ethernet switch.

KEYWORDS
Parallel processing, processor farm methodology, image processing, edge detection, noise removal, LAN

1. INTRODUCTION

Digital image processing is an ever expanding area with applications reaching out our everyday life such as medicine, space exploration, surveillance, authentication, automated industry inspection, security, and many more areas. Such applications involve different processes like image enhancement, edge detection, object detection, noise removal, color quantization, etc [1-3]. Implementing such applications on general-purpose scalar computers is easy, but, in addition to be relatively slow, it faces other drawback such as memory restrictions. The most obvious solution to meet the growing demands of image processing applications is the use of parallel (distributed) computing [4-7].

Parallel computing uses multiple computing resources to solve a time consuming computational problem faster and maintaining the same accuracy level. Furthermore, parallel computing takes advantage of non-local resources to overcome memory constraints of a single computer, cost savings by using multiple cheap computing resources, etc [8]. Fortunately, during the last three decades there has been an impressive gain in personal computer (PC) performance (speed and memory), tremendous development in computer communication technologies and internetworking methodologies, and drastic reduction in the cost of PC and communication technologies. These all are due to the tremendous advances in technology and innovations in computer and communication system architectures. It has been well recognized that to conduct

parallel computing, the computing resources can include a single advanced processor architecture and technology, a single computer with multiple processors, or an arbitrary number of computers connected by a network or a combination of the three [9].

The use of homogeneous/heterogeneous collections of low-cost computing systems (e.g., PCs, laptops, PDAs, smart phones, etc) interconnected using wired and/or wireless media forming a single logical computational resource has become a wide-spread approach to speedup up time-consuming computations. As an example of such cost-effective computing systems are the local area networks (LANs) [10].

In this paper, we first present a description of a serial, research-level, image processing application developed for edge detection in distorted images, namely, the edge detection (ED) tool. In ED tool, in order to increase the edge detection accuracy, the input images are pre-processed for noise reduction or removal using standard filters (e.g., mean, median, or Gaussian) [11], then the tool performs edge detection using one of the following well-know edge detection algorithms: Sobel, Canny, or Laplacian algorithm [12-14]. The image processing rate (λ) (number of images processed per min) that can be achieved by running the ED tool on a single computer is $1/\tau$, where τ is the image processing time. Therefore, to use the ED tool for online edge detection, the image arriving or capturing rate should be less or equal to λ, otherwise arriving images need to be buffered before processing. Consequently, the buffer size depends on τ (or λ) and the size of the arrived image. Thus, reduce the buffer size or perform a realistic online processing, $\square$ should be minimized to meet the image capturing rate.

In this paper, we present our work on implementing and evaluating the performance of a LAN in speeding up image processing applications. To demonstrate that we develop a parallel version of the ED tool to run concurrently on number of computers interconnected into a LAN. The parallel version is called LAN-based edge detection (LANED). The philosophy adopted and the parallelization methodology used in transferring ED tool to run on the LAN, are discussed. The LANED tool uses the Java parallel virtual machine (JPVM) to exchange data between the different PCs on the LAN [15-17]. For equivalent calculations, the computation times on a single PC and a LAN of various number of PCs, are estimated, and the resulting speedup and parallelization efficiency computed. Furthermore, in this paper, we investigate the effect of a number of parameters, such as: size of the LAN, number of images, and size of convolution function, on the LAN performance.

This section presents an introduction to the main topics, objectives, and outcome of this paper. Section 2 presents a literature review that summarizes the most recent and related work. Section 3 briefly describes the ED tool. The parallel programming methodologies are described in Section 4. The parallel implementation of the ED tools is presented in Section 5. Some performance measures are defined in Section 6, while the results and discussions are given in Section 7. Finally, in Section 8, based on the results obtained, a number of conclusions are drawn and recommendations for future work are pointed-out.

2. LITERATURES REVIEW

In this section, we review some of the most recent and related work, which include a number of parallel and distributed models for image processing applications on different types of parallel computer architectures, and some dedicated parallel and distributed models that run on LAN-Based systems.

H. Fatemi et. al. [5] presented and evaluated a method for introducing parallelism into an image processing application. The method is based on algorithmic skeletons for low, medium and high

level image processing operations. They provided an easy-to-use parallel programming interface. The approach identified number of skeletons for parallel processing of low-level, intermediate-level and high-level image processing operations. Each skeleton can be executed on a set of processors. From this set of processors, a host processor is selected to split and distribute the image to the other processors. The other processors from the set receive a part of the image and the image operation which should be applied to it. Then, the computation takes place and the result is sent back to the host processor. To evaluate this approach, face recognition was implemented twice on a highly parallel processing platform, namely, the IMAP-board, once via skeletons, and once directly and highly optimized. It was demonstrated that the skeleton approach is extremely convenient from a programmer's point of view, while the performance penalty of using skeletons is well below 10% in the case study.

W. Caarls et. al. [18] designed asynchronous remote procedure call (RPC) system to exploit low-level image processing operation task-level parallelism to be used for algorithmic skeletons. The system was programmed in C language, divided into number of image processing operations, and applied these using function calls. They implemented a double threshold edge detection algorithm on a prototype architecture consisting of XETAL 16 MHz 320-PE SIMD (single instruction multiple data) processor and a TriMedia 180 MHz 5-issue VLIW processor. The result showed that the overhead of running the RPC system is around 8%, but decreasing processing time about 42%. Their result also showed that the system can achieve a significant speedup by using SIMD processor for low-level vision processing.

H. Kelash et. al. [19] presented parallel processing using multi-agent system which can be structured into application interface that allows to call particular operators or to pass image processing operation for parallelization. In their system, each agent has a very simple behavior which allows it to take a decision such as find out an edge, or region, etc., according to its position in the image and to the information enclosed in it. The system provides an environment for developing and processing image operations within distributed system. Data parallelism was implanted in this system, where all processing elements (PEs) receive commands from a central control processor. The system uses the CxC language, and applies Sobel and Laplace operators using different data which can be parallelized using array controller of processors where one processor associated with one pixel. They compared between their multi-agent system and the sequential execution using MATLAB. They found that the speedup factor is increasing when using multi-agent system as the size of images increases.

D. V. Rao et. al. [20] addressed the implementation of image processing algorithms like image filtering, image smoothing and edge detection on field programmable gate array (FPGA) using Handle-C language which is a C-based language that can provide direct implementation of hardware from the C-based language description of the system. The design was implemented on RC1000-PP Xilinx Vertex-E FPGA based hardware. The results from this design used operations for the image processing algorithms on a 256x256 size grayscale of Lena image show that the speed of this FPGA solution for the image processing algorithms was approximately 15 times faster than the software implementation in C language.

F. Schurz and D. Fey [21] presented a parallel processor architecture based on small PEs in a FPGA. Their architecture is able to detect and process multiple separated objects simultaneously in image which is divided into partitions and handled one by one to keep the whole design small. The architecture is using SIMD approach, which means that the same operations are carried out in parallel on each image pixel. The PEs in this design are connected through a NEWS network and controlled by a central unit. Their design is programmable using assembler language. This approach designed to be small and cheap and fast possibility for industrial image processing. The results for this design, in a VGA resolution approximately one and half million clocks, were used and 66 images can be processed at 100 MHz, which leads to a performance of 20 MPixel/s.

F. Baldacci and P. Desbarats [22] presented a parallel algorithm for 3D split and merge segmentation using topological and structuring with an oriented boundary graph image processing. They used multiprocessor systems and non-uniform memory access (NUMA) architecture. The algorithm was tested in two machines. First machine was equipped with two Intel Xeon Quad core at 2.33 GHz, and the other was equipped with 8 AMD Opteron Dual core at 1.8 GHz with NUMA architecture. They used two medical images in test: one image with size 256x256x256 voxels and the other with 512x512x475 ovxels size. The results studied the execution time and showed that the NUMA architecture was two time slower than the other one, and using 16 threads was slower than using 8 threads.

A. Bevilacqua [23] introduced a model to obtained efficient load balancing for data parallel applications based on dynamic data assignment running on a heterogeneous cluster of workstations. The model was referred to the working-manager model, which aims was to maximize the performance of loosely coupled systems. It is essential to minimize the idle time of each process and ensure the balancing of processes workload. The cluster used consists of four workstations, connected to a LAN by a 100Mbps Ethernet, except for workstation 3, which was connected by 10Mbps adapter. The results showed that the efficiency was over 90%.

J. A. Gallud et. al. [24] presented a workbench called distributed processing of remotely sensed imagery (DIPORSI). It was developed to provide a framework for the distributed processing of Landsat images using a cluster of NT workstations connected by Ethernet network using the message passing interface (MPI) standard. The distributed machine in their model is composed of 8 P-II 333 MHz with 32 MB of RAM running windows NT Workstation v4.0, and the nodes were linked using a 10 Mbps Ethernet. The results showed that a reduction of 400% in the execution time for a moderate number of nodes can be achieved. The results also showed that a near linear speedup for large image size can be achieved.

H. S. Bhatt et. al. [25] developed an environment over a network of VAX/AMS and UNIX for distributed image processing. They redesigned and generalize DEDIP (development environment for distributed image processing) to make it more user-friendly and truly heterogeneous, using Java and Web technology, therefore, they referred to the new environment as WebDEDIP, which has three tier architectures: GUI, DEDIP server, and agents, instead of master-slave one. The functionality and efficiency of the WebDEDIP was tested using Microsoft NT as host and IRIS workstations as a slave. IIS 4 was used as a Web server and the front-end GUI was tested on two most popular browsers IE and Netscape. The model was used by 15 scientists for development and operationalization of 10 distributed image processing applications for Indian remote sensing (IRS) satellite. The efficiency was as high as 90-95%.

C. Nicolescu and P. Jonker [26] presented a data and task parallel low-level image processing environment for distributed memory system. They designed an approach of adding data and task parallelism to an image processing library using algorithmic skeletons and the image application task graph (IATG). They used a distributed system which consists of a cluster of Pentium Pro/200 MHz PCs with 64MB RAM running Linux, and connected through Myrinet in a 3D-mesh topology with dimension order routing. The code was written using C and MPI message passing library and the multi-baseline stereo vision algorithm is an example used in their system. They concluded that the speedup in data and task parallel approach was more efficient than the speedup in data parallel approach only.

Z. Qiu et. al. [27] developed fast parallel stereo matching parallel algorithm on home-based software DSM JIAJIA. A cluster of 8 P-II PCs connected by a 100 Mbps switched Ethernet was used. The stereo images were divided into 8 parts. Each PC carried out the matching task of one parts of stereo image. The speedup ratio is near the ideal linearity speedup ratio. The speedup of finding corresponding points reaches 3200 pair/second, when 8 PCs were used.

J. O'Connell and P. Caccetta [28] presented an algorithm used for time series classification of remotely sensed image data which is spatial/temporal algorithm. Their approach used homogeneous and heterogeneous clusters of computers for reducing computational time using the MPI standard library. The parallel algorithm distributes each line of the input probability images to a number of slave nodes with I/O performed by one master node. Slave nodes then perform the necessary processing tasks and send the output back to the master. The parallel algorithm implemented on two clusters, an ad hoc cluster and the dedicated cluster. The ad hoc cluster used 13 office Wintel machines. All machines were P-IV between 1.6 GHz and 3.6 GHz and of signal dual and quad CPU connected via 100 Mbps Ethernet. The MPICH implementation was used in this cluster. The results showed that the efficiency in an ad hoc cluster at least 67% in homogenous CPUs, but the efficiency in the dedicated cluster was about 86.2% (speedup 7.76) in 9 CPU, and 85.43% (speedup 41.86) in 49 CPU.

A. K. Manjunathachari and K. SatyaPrasad [7] designed approach to solve the convolution filter by using simultaneous multi-threading (SMT), processing buffer (PB), and simulated in a standard LAN environment. Their approach presented a method to the bifurcation of image processing application into three fundamental layers (resource layer, linking layer and application layer), which are isolated based on processor requirements and their functionality. The parallelism was enhanced by adding the concepts of SMT over the processor for redundancy the transition delay in parallel computing image processing application. Results showed that for a large number of processing units, speedup is close to linear, and also speedup characteristics were identical when the same number of templates was used in the matching process. The approach used two different implementation methods for parallel image convolution. The first method was the direct convolution method which has less communication load than the other method, which was 2D fast Fourier transform (FFT) in a Fourier domain. Direct convolution method's scalability slightly decreased as kernel size got smaller but hardly affected by image size. The other method's scalability decreased as image size got smaller and never affected by kernel size.

A. Paz et. al. [29] developed several parallel algorithms for target detection in hyper-spectral imagery. They developed four algorithms for target and anomaly detection in hyper-spectral images, these algorithms are: the automatic target generation process (ATGP), an unsupervised fully-constrained least squares (UFCLS) algorithm, an iterative error analysis (IEA) algorithm, and RX algorithm which developed by Reed and Xiaoli for anomaly detection. The problem in these algorithms were computational very expensive. They solved the computational problem by developed four computationally efficient parallel implementations, a parallel ATGP (P-ATGP) algorithm, a parallel UFCLS (P-UFCLS) algorithm, a parallel anomaly detector (P-RXD) and a parallel MORPHological target detection algorithm (P-MORPH). In all algorithms they used a data-driven partition strategy tested on a hyper-spectral image scene collected by the AVIRIS instrument. The full data in the experiment consists of 2133x512 pixels, 224 spectral bands and total size about 900 MB. They used a single processor of a Beowulf cluster with 256 processors called Thunderhead and available at NASA's Goddard Space Flight Center. The results showed that the computation time of the parallel algorithms was more efficient of the computation time in sequential algorithms.

3. THE EDGE DETECTION (ED) TOOL

This section presents a description of the edge detection (ED) tool, which is a software tool especially developed for edge detection in distorted images. In order to ensure accurate edge detection with ED tool, we first reduce or ultimately remove noise from images, then apply edge detection technique [2]. The current version of ED tool can be configured to use one of the following noise-removal filters: mean, median, or Gaussian filter [11]. For edge detection, one of the following techniques can be used: Sobel, Canny, or Laplacian technique [12-14]. Sobel and

Canny techniques detect edges by looking for the maximum and minimum in the first derivative of the image; while the Laplacian technique searches for zero-crossings in the second derivative of the image in order to find edges.

Both noise reduction and edge detection algorithms used in ED tool are based on a simple mathematical function, it is the convolution function, which is a multiplication of two arrays, the image and the kernel arrays. The kernel array is usually much smaller than the image array, and is also two dimensional (although it may be just a single pixel thick). The kernel array could be of different size; such as 3×3, 5×5, 7×7, etc. If the image has M rows and N columns, and the kernel has m rows and n columns then the convolution function is written as [2, 11]:

$$H(i,j) = \sum_{k=1}^{m} \sum_{l=1}^{n} G(i+k-1, j+l-1) K(k,l) \qquad (1)$$

Where i runs from 1 to M-m+1 and j runs from 1 to N-n+1. The function moves the kernel K through the image G pixel by pixel, at each point the overlapping pixels in the image and kernel arrays are multiplied and then summed to get new value for the pixel. Convolution function is a very complex operation that requires huge computation power. To calculate a pixel for a 3x3 kernel, there are 9 multiplications per image pixel, if the input image is 1024×1024, then the convolution function needs more than 9×10^6 multiplications.

4. METHODOLOGIES FOR PARALLEL PROGRAMMING

Parallel computing is the design, implementation, and tuning of computer programs to take advantage of parallel computing systems [8, 9]. It focuses on partitioning the overall problem into separate tasks (processes and data), allocating tasks to processors and synchronizing the tasks to get meaningful results. For most applications, there are three common broad methodologies for parallel programming; these are: processor-farm or event methodology, geometric methodology, and arithmetic or algebraic methodology [30].

In this paper, the processor farm methodology [31, 32] is used in porting the serial code to run on the LAN; therefore, we shall discuss it in details next. In geometric methodology, each processor executes more or less the same program but the data is distributed in a manner which requires extensive communication between the processors, for example, each processor might be used to simulate one part or more of a large system of similar objects interacting with each other. While in arithmetic or algebraic methodology this methodology, the whole algorithm is split into a number of sections, each of which is assigned to one processor, but data relating the whole system flows through each processor like a production line. Thus, complicated and extensive communication is required in transferring the data from one processor to another. Further details on these two methodologies can be found in [8, 9, 30].

4.1 Processor farm methodology

The processor-farm is may be considered as the simplest methodology in which each processor executes the same program independently from all other processors, each operates on different part of the total data [31, 32]. Therefore, this methodology is very suitable for applications where the same process has to be applied to a number of independent data sets. It allows the same sequential program to be implemented with minor modifications to create two different versions of the program, namely, the master and the slave programs. The processor that runs the master program is called the master processor, while the processor that runs the slave program is called the slave processor. Usually, exactly the same slave program is loaded and allowed to run on more than one processor concurrently performing its calculations on different data. This last

feature requires that enough memory is available to accommodate the whole program on each processor.

In this methodology, the standard relationship between the master and the slaves programs can be summarized as follows:

1. The master reads-in the input data and perform any require preliminary computations.
2. Send the data to the appropriate slave.
3. After completion of the computations, the slave sends the output results back to the master processor.
4. The master processor performs any post-processing computations and then presents the final outputs.

It is clear from the above relationship that the processor-farm involves no inter-processor communications other than in forwarding data/results between the master and the slaves, once for all at the beginning and the end of the computation. Furthermore, slaves are only allowed to communicate with the master.

5. THE LAN-BASED EDGE DETECTION (LANED) TOOL

This section presents a description of the parallel implementation of sequential ED tool on a LAN-based computing system to speed up the image processing computations by efficiently utilizing the relatively high computational power of the LAN to provide a cost-effective solution. The performance of a LAN-based computing system depends on a number of factors, these include:

1. Number of PCs used to perform the computational task concurrently.
2. Speed of each individual PC on the LAN.
3. Speed of the communication channels.
4. Efficiency of the message passing library.
5. Parallel programming model that is used in porting the computational task to the parallel system.

The parallel methodology that is going to be used in transferring (parallelizing) the serial computation of ED tool is based on the processor-farm strategy. In which the parallelized version of the code is developed in two versions, one version is developed to run as on a master PC acting as a server, and the other version is to run as one or more slave computer(s) (client(s)). More than one slave is usually loaded and run concurrently on different processor performing its calculation on different data (images) (also refer to as multiple instructions multiple data (MIMD) architecture) [33]. This form of paradigm involves no inter-processor communication and slaves are only allowed to communicate with the master. The relationship between the master-slave is summarized as follows:

1. The master reads-in the input data (images) and perform any require preliminary computations.

2. The master sends necessary data to one or more appropriate slave(s).

3. The slave(s), after completion of the computations, send(s) the results back to the master.

4. The master performs any post-processing computations and then presents the final outputs.

It is clear from the above relationship that the master is idle while waiting for the slaves to complete their computations. Therefore, in order to utilize the master to perform some useful computations, instead of being idle while waiting, it is used to run as a slave. However, since the master starts performing its computation after sending all data to all slaves (after all slaves start), then it is expected that all of them will finish first, and they have to wait until the master complete its computations before sending their results back to the master. In order to avoid this conflict, the size of the task assigned to the master should be less, so that it can finish before the slaves complete their computations, and be ready to receive their results.

The LANED tool can efficiently process many images at a time. These images may be obtained from on-line (real-time) or offline image sources. On-line image sources include capturing devices, such as: digital camera, satellite images, etc. Off-line image processing means those images which are captured, stored to be processed in a later time, such as internet images. Furthermore, the LANED tool can be configured to accommodate more than PC as a mater processor. The LANED tool is implemented using the JPVM environment as message passing tools between PCs on the LAN, and the image processing applications in Java language.
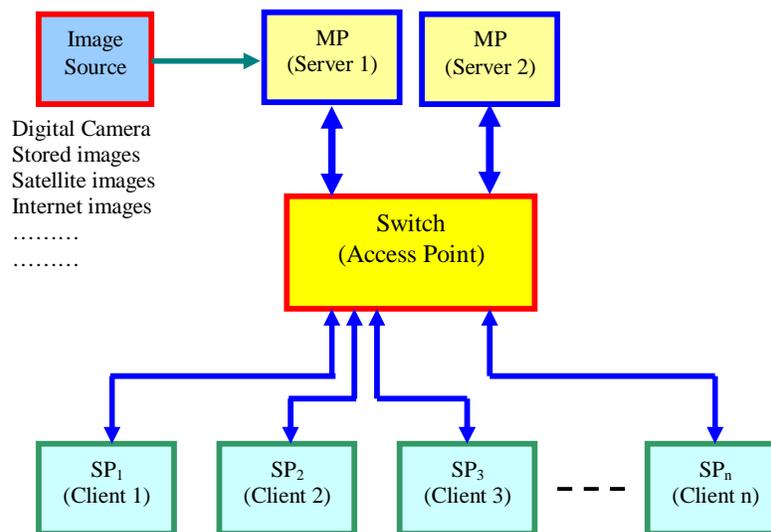

Figure 1. System architecture and the data flow of the LANED tool.

5.1. The Data Communication Library

There are many message passing libraries in use between the different PCs on the LAN, such as: message-passing interface (MPI) [24, 31], parallel virtual machine (PVM) [34], and Java parallel virtual machine (JPVM) [15-17].

5.1.1   Parallel virtual machine (PVM)

The PVM is a software system that permits a collection of many heterogeneous computers that networked together to be one large computer working in parallel mode [34]. The PVM is designed to link computing resources together and provided users with a transparent efficient parallel platform for running their computer applications. The PVM transparently and independently handles all message routing data conversion and task scheduling across a network of incompatible computer architectures. Therefore, it is used in many sites all over the world to solve important problems in scientific, engineering, industrial and in medical applications. The

PVM system can be used to run different computers in parallel mode (concurrently), and it is designed to have many important features and capabilities, such as:

1. Reduce the cost to solve problems.
2. Reduce the contention for resources.
3. More effective implementations of an application.
4. Make the parallel programming in a heterogeneous collection of processors straightforward.

5.1.2   Java parallel virtual machine (JPVM)

The Java language and its libraries and environment provide a powerful and flexible platform for programming computer clusters. Java tools enable experimentation in both management aspects as well as performance aspects of cluster systems [15-17].

The JPVM is a PVM like library of object classes implemented in and for use with the Java programming language. The library supports an interface similar to C and FORTRAN interfaces provided by the PVM system, but with syntax and semantics enhancements afforded by Java and better matched to Java programming styles.

The JPVM is a combination of both ease of programming inherited from Java and high performance through parallelism inherited from the PVM. The JPVM library is software used for message passing in distributed memory MIMD LAN-Based parallel computing system. The JPVM has many features not found in standard PVM, such as [15-17]:

1. JPVM is thread safety; it can control multiple Java threads inside a single JPVM task.

2. Standard PVM has single communication end-points for every task, but the JPVM can create a new task within a process every time, so it has multiple communication end-points for each task.

3. JPVM code can be maintained much simpler than the PVM across heterogeneous machine.

4. JPVM has default-case direct message routing.

For as mention features of Java language and JPVM; the LANED tool uses the JPVM as a parallel environment.

As in the PVM, the programmer decomposes the problem to be solved into a set of cooperating sequential task implementations. These sequential tasks execute on a collection of available processors and invoke special library routines to control the creation of additional tasks and to pass messages among tasks. In JPVM, task implementations are coded in Java, and support for task creation and message passing is provided by the JPVM library.

The architecture of the JPVM is similar to architecture of the PVM, which is consisting of the daemon, the console and the interface library functions. The JPVM library routines require run-time support during execution in the form of a set of JPVM daemon processes running on the available collection of processors. The console can start in any processors in the network. The JPVM console can be used to list the hosts available to the system and the JPVM tasks running in the system.

Tasks in the JPVM environments are process-based; however the communications are using transfer control protocol (TCP) sockets through the network [10]. Figure 2 outlines the JPVM architecture.
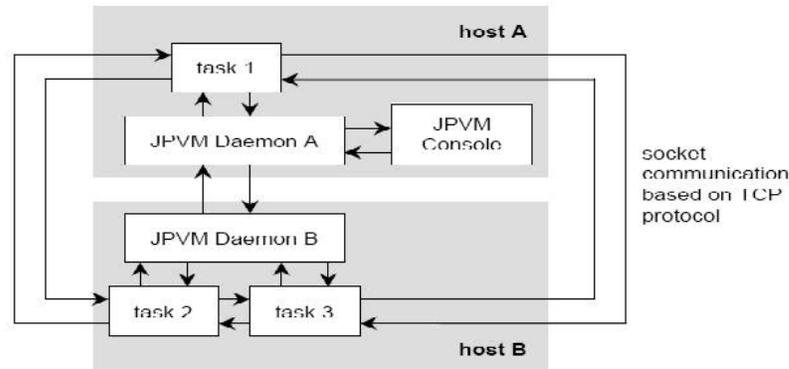

Figure 2. JPVM architecture[17].

5.2. Implementation of the LANED Tool

First the JPVM platform must starting by run *jpvmDaemon.java* program in all computers in the LAN-based system, then run *jpvmConsole.java* in one computer as a master computer. The master controls the message passing techniques in the network. The master PC starts to capture or input sequence of noisy images from devices or files, these images may have similar or different sizes. Task creation start in master program by using *jpvm.pvm_spawn( )* method, which has number of slaves and the java class program for slave.

When the master has number of images it is start to send these images between slaves by sending the same number of images for each slave on the LAN, the distribution of images depends on the total number of images and number of slaves.

The master does not send all images at the same time; it is sends images one by one to each slave using a *for-loop* to prevent slaves from being idle while waiting until first slaves receive their images, the images are stored in buffers. Then each slave reads one image from the buffer at a time and starts processing the images sequentially until it processes all images sent by the master.

At this time when the master finishes send all images to slaves it starts processing number of images to save time and to be not idle while waiting till the completion of slaves. As we explained above, for more efficient load balancing [23], the number of images allocated for the master must be less than number of images allocated for the slaves. The number of images allocated for the master depends on number of images allocated for each slaves on the LAN. When slaves finish processing their images, they start sending processed images back to the master. The master starts receiving all images from slaves and then the master output the resultant images.

6. PERFORMANCE MEASURES

In order to measure the performance of a parallel algorithm running on a parallel machine two performance measures are usually considered, these are:

(i)   Speedup factor (*S*)

Speedup factor (S) is defined as the ratio between the time required to perform a particular computation on a sequential mode machine (T_s) and the time required to perform an equivalent computation on a parallel mode machine (T_p)., and it is calculated as $S=T_s/T_p$ [8, 9]. For a LAN, T_s represents the time required to perform the computation on a single PC, and T_p is the time required to perform the computation on all active PCs including both the master and slaves that are participating in the computations.

Ideally, the maximum speedup that can be achieved is equal to the number of active PCs on the LAN (C) (master plus the number of running slaves). However, there are several factors that limit and prevent the speedup from reaching its maximum value, such as:

1. Load balancing when not all computers perform useful computation all the time, and some of the computers may be left simply idle for a period of time during the computation.
2. Software overhead due to the extra computation may be required in the parallel version of the code not appearing in the sequential version, for example, to recomputed constants locally.
3. Communication time for data and messages exchange among the processors.

(ii)   Parallelization efficiency (E)

Another factor of interest is the parallelization efficiency (E) [8, 9], which is defined as the ratio between S and C (i.e., $E=100\times S/C$). E can also be defined as the actual computation time (T_{comp}) divided by the total computation and communication times (T_p), which represents the sum of T_{comp}, communication time (T_{comm}), and other overheads (T_{over}). Accordingly, E can be given as: $E=100\times T_{comp}/(T_{comp}+T_{comm}+T_{over})$. T_{over} is very small compared to T_{comp} and T_{comm}, thus E can be expressed as: $E=100\times T_{comp}/(T_{comp}+T_{comm})$. It is clear from the above two equations that E depends on the amount of time that is spent on communication or on the ratio R between the communication and computation times. The maximum efficiency can be achieved when T_{comm} (i.e., R) approaches zero.

7. RESULTS AND DISCUSSIONS

In order to evaluate the performance of the LANED tool, a number of edge detection applications were performed on a LAN consisting of 5 PCs interconnected through an Ethernet 10/100 Mbps switch. The PCs are Acer verition GT series, Intel (R), Pentium IV processor with 2.8 GHz speed and running Windows XP operating system. The image processing analysis performed by the LANED tool includes noise removal using median filter and edge detection using Sobel algorithm. The input to these applications is listed in Table 1.

Table 1. Input data

Parameters	Value/Type
Edge detection technique	Sobol
Filter type	Median
Type of noise	Impulsive noise
Image size	256x256
Number of images	100, 200, 300, 400, 500
Kernel size	3x3, 5x5

The results for the computation time, *S*, and *E* are presented in Table 2. Moreover, the variation of *S* and *E* are shown in Figures 3 and 4, respectively. These results can be used as a roadmap for a number of outcomes about the performance of a LAN-based image processing system. The main outcomes can be summarized as follows:

1. For an image of 256×256 pixel, the average image processing time on our PCs is 0.125 sec for 3×3 kernel size and 0.365 sec for 5×5 kernel size. So that the average pixel processing time is 1.91 μsec for 3×3 kernel size and 5.57 μsec for 5×5 kernel size.

2. For a fixed *m* and *k*, *E* decreases with increasing *n*, because as *n* increases, T_{comm} becomes the dominant part as compared to T_{comp}. This is due to the fact that a LAN has high message start-up latencies and low bandwidths. Furthermore, as *n* increases, the number of images that needs to be sends by the master to the slaves is increasing.

3. For fixed *n* and *m*, *E* increases when *k* increases from 3×3 to 5×5; because as we discussed earlier, T_{comp} is increased by triple while T_{comm} should remained unchanged.

4. According to our analysis, for the ranges of *n*, *k*, and *m* investigated in this work, for fixed *n* and *k*, *m* has no or insignificant effect on *E*, because as *m* increases both T_{comp} and T_{comm} are increased keeping *E* unchanged.

The LANED tool achieved a *S* of over1.6 (E~30%) on a LAN of 5 PCs when 3×3 kernel size used and a *S* of over 3.0 (E~60%) for 5×5 kernel size. From this last point, we can realize that as the average image or pixel processing time increases, then *S* and *E* will be increased. Thus, higher parallelization efficiency is expected for 7×7 kernel size. In general, we can conclude that the processor-farm methodology is better suited to image processing applications that relatively has no communication overheads and require high average pixel processing time.

Table 2. Performance of the LANED tool running on a LAN of various number PCs (*n* 5)
(Results for 3×3 and 5×5 convolution function computation)

| No. of PCs	m=100		m=200		m=300		m=400		m=500	
	K									
	3x3	5x5	3x3	5x5	3x3	5x5	3x3	5x5	3x3	5x5
CPU time (sec)										
1	12.50	36.6	24.91	73.1	37.20	109.8	49.66	146.0	62.03	182.5
2	9.31	21.7	17.88	42.4	29.88	63.1	41.58	83.3	52.92	104.9
3	8.62	16.6	17.20	31.9	25.71	47.6	35.00	63.6	43.17	79.3
4	8.30	14.2	15.76	27.3	23.79	40.7	31.62	52.7	38.91	66.0
5	7.67	12.4	15.06	24.2	22.30	36.2	29.89	48.1	36.88	59.5
Speedup factor (S)										
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	1.34	1.69	1.39	1.72	1.24	1.74	1.19	1.75	1.17	1.74
3	1.45	2.20	1.45	2.29	1.45	2.31	1.42	2.30	1.44	2.30
4	1.51	2.58	1.58	2.68	1.56	2.70	1.57	2.77	1.59	2.77
5	1.63	2.95	1.65	3.02	1.67	3.03	1.66	3.04	1.68	3.07
Parallelization efficiency (E) (%)										
1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
2	67.1	84.3	69.7	86.2	62.2	87.0	59.7	87.6	58.6	87.0
3	48.4	73.5	48.3	76.4	48.2	76.9	47.3	76.5	47.9	76.7
4	37.7	64.4	39.5	66.9	39.1	67.4	39.3	69.3	39.9	69.1
5	32.6	59.0	33.1	60.4	33.4	60.7	33.2	60.7	33.6	61.3
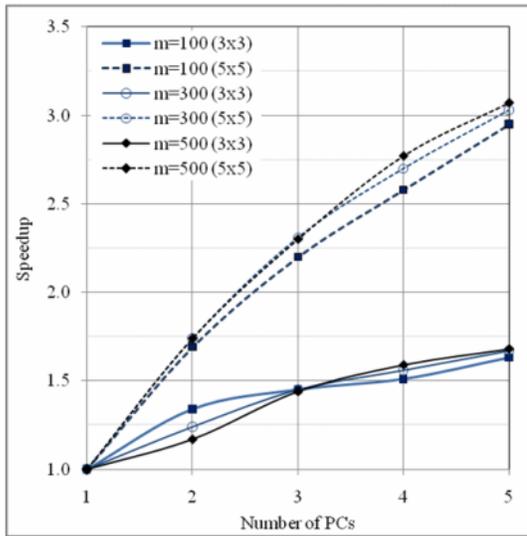
Figure 3. Variation of *S* against *n*.
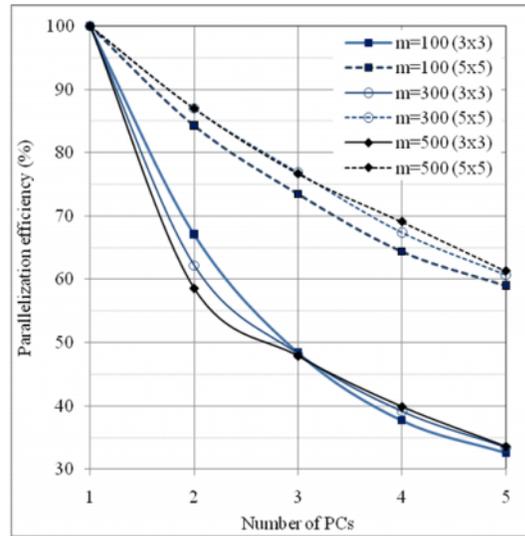
Figure 4. Variation of *E* against *n*.

8. CONCLUSIONS

The main conclusions can be summarizes as follows:

1. A standard Ethernet LAN can be successfully used as a cost-effective, efficient, and reliable computing platform to speedup image processing computations, subject to the development of an efficient parallel (distributed) implementation model.

2. The processor-farm methodology is a simple and easy to implant and configure to meet application needs, where, in this methodology, the same version of the code is running on all slaves, with little variation required for the master.

3. The performance depends on the number of PCs on the LAN, pixel average processing and communication times. The results in this paper demonstrated that the processor-farm methodology provided an excellent parallelization efficiency of over 60% on 5 PCs network for moderate average pixel processing time. However, this can be improved further by introducing an exceptional load balance across the network.

For future work, it is highly recommended to carry on with a number of research projects to implement and compare the performance for larger LAN, evaluate the performance of the tool for various image size, different network technologies and protocols (such as: 100 Mbps (IEEE 802.3u), 1 Gbps (IEEE 802.3z and IEEE 802.3ab), 10 Gbps (IEEE 802.3ae, IEEE 802.3ak, IEEE 802.3an), etc., different LAN network topologies, wireless LAN utilizing the IEEE 802.11 protocol in access point and ad-hoc configurations. Also, develop a version of LANED utilizing other parallel programming methodologies (algorithmic model or geometric model).

REFERENCES

[1] Rafael C. Gonzalez and Richard E. Woods. Digital Image Processing. Prentice-Hall Inc., 2nd Edition, 2002.

[2]   P. Civicioglu and M. Alci. Edge Detection of Highly Distorted Image Suffering from Impulsive Noise. International Journal of Electronics and Communications, Vol. 58, No. 6, pp. 413-419, 2004.

[3]   Mohamed Roushdy. Comparative Study of Edge Detection Algorithms Applying on the Grayscale Noisy Image Using Morphological Filter. ICGST International Journal on Graphics, Vision and Image Processing (GVIP), Vol. 6, Issue 4, pp. 17-23, 2006.I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in *Magnetism*, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4]   Thomas Bräunl. Tutorial in Data Parallel Image Processing. Australian Journal of Intelligent Information Processing Systems (AJIIPS), Vol. 6, No. 3, pp. 164-174, 2001.

[5]   Hamed Fatemi, Henk Corporaal, Twan Basten, Pieter Jonker, and Richard Kleihorst. Implementing Face Recognition Using a Parallel Image Processing Environment Based on Algorithmic Skeletons. Proceedings of the 10th Annual Conference of the Advanced School for Computing and Imaging, pp. 351-357, 2004.

[6]   A. Clematis, D. D'Agostino, and A. Galizia. A Parallel Image Processing Server for Distributed Applications. John von Neumann Institute for Computing, NIC Series, Vol. 33, pp. 607-614, 2006.

[7]   A. K. Manjunathachari and K. Satya Prasad. Implementation of Image Processing Operations Using Simultaneous Multithreading and Buffer Processing. ICGST International Journal on Graphics, Vision and Image Processing (GVIP), Vol. 6, Issue 3, pp. 47-53, 2006.

[8]   Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. Introduction to Parallel Computing. Pearson, Addison Wesley, 2nd Edition, 2002.

[9]   Fayez Gebali. Algorithms and Parallel Computing. John Wiley & Sons, Inc. 2001.

[10]   Behrouz A. Forouzan. Data Communications and Networking. McGraw-Hill, 4th Edition, 2007.

[11]   Bob Fisher, Simon Perkins, Ashley Walker and Erik Wolfart. Hypermedia Image Processing Reference. Department of Artificial Intelligence, University of Edinburgh, 1994.

[12]   O. R. Vincent and O. Folorunso. A Descriptive Algorithm for Sobel Image Edge Detection. Proceedings of Informing Science & IT Education Conference (InSITE), Macon State College, Georgia, USA, June 12-15, 2009.

[13]   J. F. Canny .A Computational Approach to Edge Detection. IEEE Trans Pattern Analysis and Machine Intelligence, Vol. 8, Issue 6, pp. 679-698, 1986.

[14]   Shen-Chuan Tai and Shih-Ming Yang. A Fast Method for Image Noise Estimation Using Laplacian Operator and Adaptive Edge Detection. Proceedings of the 3rd International Symposium on Communications, Control, and Signal Processing (ISCCSP'08), pp. 1077-1081, Malta, March 12-14, 2008.

[15]   Adam Ferrari. JPVM: Network Parallel Computing in Java. Concurrency: Practice and Experience. Vol. 10, No. 11-13, pp. 985-992, 1998.

[16]   Narendar Yalamanchilli and William Cohen. Communication Performance of Java-based Parallel Virtual Machines. Department of Electrical and Computer Engineering, University of Alabama in Huntsville, USA, 1998.

[17]   Bu-Sung Lee, Yan Gu, Wentong Cai, and Alfred Heng. Performance Evaluation of JPVM. Journal Parallel Processing Letters, Vol. 9, No. 3, pp. 401- 410, 1999.

[18]   Wouter Caarls, Pieter Jonker, and Henk Corporaal. Skeletons and Asynchronous RPC for Embedded Data and Task Parallel Image Processing. IAPR Conference on Machine Vision Application (MVA'05) (pp. 16-18). Tsukuba Science City, Japan, May 16-18, 2005.

[19] H. Kelash, M. Zaki Gamal El_Dein, and N. Kamel. Agent Distribution Based Systems for Parallel Image Processing. ICGST International Journal on Graphics, Vision and Image Processing (GVIP), Vol. 6: Special Issue on Applicable Image Processing Techniques, pp. 87-92. 2006.

[20] Daggu Venkateshwar Rao, Shruti Patil, Naveen Anne Babu, and V. Muthukumar. Implementation and Evaluation of Image Processing Algorithms on Reconfigurable Architecture Using C-Based Hardware Descriptive Languages. International Journal of Theoretical and Applied Computer Sciences, Vol. 1, No. 1, pp. 9–34, 2006.

[21] Frank Schurz and Dietmar Fey. A Programmable Parallel Processor Architecture in FPGAs for Image Processing Sensors. Proceedings of the Integrated Design and Process Technology (IDPT'07) Conference, pp. 30-35. Antalya, Turkey, June 3-8, 2007.

[22] Fabien Baldacci, Achille Braquelaire, Pascal Desbarats, and Jean-Philippe Domenger. 3D Image Topological Structuring with an Oriented Boundary Graph for Split and Merge Segmentation. Proceedings of the 14th IAPR International Conference on Discrete Geometry for Computer Imagery (DGCI'08) (pp. 541–552). Lyon, France, April 16-18, 2008.

Authors

Ghassan F. Issa (gissa@uop.edu.jo). He received his B.E.T degree in Electronic Engineering from the University of Toledo, Ohio, in 1983, and B.S.EE in Computer Engineering from Tri-State University, Indiana in 1984. He received his M.S. and Ph.D. in Computer Science from Old Dominion University, Virginia, in 1987 and 1992 respectively. He was a faculty member and department chair of Computer Science at Pennsylvania College of Technology (Penn State) from 1992–1995. He also served as faculty member and the dean of Computer Science at the Applied Science University in Amman, Jordan from 1995-2007. Currently he is an associate professor and the dean of faculty of information technology at Petra University in Amman, Jordan. His research interest covers block cipher, and authentication.

Hussein Al-Bahadili (hbahadili@uop.edu.jo) is an associate professor at Petra University. He received his PhD and M.Sc degrees from University of London (Queen Mary College) in 1991 and 1988. He received his B.Sc in Engineering from the University of Baghdad in 1986. He is a visiting researcher at the Centre of Wireless Networks and Communications (WNCC) at the School of Engineering, University of Brunel (UK). He has published many papers in different fields of science and engineering in numerous leading scholarly and practitioner journals, and presented at leading world-level scholarly conferences. He has published three chapters in prestigious books in IT and Simulations. He is also a reviewer for a number of books, and currently, he is an editor of a book on Simulation in Computer Network Design and Modeling: Use and Analysis. His research interests include computer networks design and architecture, routing protocols optimizations, parallel and distributed computing, cryptography and network security, data compression, software and Web engineering.

Shakir M. Hussain (shussain@uop.edu.jo) He received his B.A. degree in statistics from University of Al-Mustansiriyah, Iraq, in 1976 and M.Sc. degree in Computing and Information Science from Oklahoma State University, USA, in 1984. In 1997 he received his Ph.D. degree in Computer Science from University of Technology, Iraq. From 1997 to 2008 he was a faculty member at Applied Science University, Jordan. Currently, he is an associate professor, dean assistant of faculty of information technology, and department chair of computer Science at the Petra University, Jordan. His research interest covers block cipher, key generation, authentication, and data compression. He is a member of ACM.