IMPLEMENTATION OF PARALLEL ADAPTIVE STACK FILTER USING PVM

Youngrock Yoon, Hyeran Byun, Yilbyung Lee

Computer Science Department
Yonsei University
Seoul, Korea
lennon@csai.yonsei.ac.kr

Jisang Yoo

Electronics Department
Hallym University
Chuncheon, Korea
jsyoo@sun.hallym.ac.kr

ABSTRACT

Stack filter showed good performance for signal restoration and noise reduction especially for impulsive noises, but required too much resource. Parallel adaptive stack filtering algorithm was developed to overcome this problem and implemented on parallel architecture. We implemented this algorithm using new heterogeneous parallel computing environment known as parallel virtual machine(PVM), using a well-known master-slave scheme. Load balancing, which is another important factor of heterogeneous computing, was used to balancing workloads of each host which is used as a separated parallel processor. It showed a better performance than previous implementation, when it was applied to a big image with large window size using reasonable number of hosts as the parallel processor.

1. INTRODUCTION

Among the many non-linear filters, stack filter has superiority due to its known optimal training algorithm using mean square error criterion. However, because of its serial nature, implementing stack filter required huge amount of time and computational resource. Fast parallel adaptive stack filter was developed to reduce computational time [1], and was implemented on the machine which has parallel architecture.

An emerging tool of parallel scheme, known as parallel virtual machine(PVM) has enabled many serial-natured network-available architecture to work as a parallel machine [2]. Architectures, which are composed of many workstations and PCs working with local area network, are common in most computing environment around the world recently.

We developed a new implementation of parallel adaptive stack filter using PVM, which used master-slave scheme of PVM, and simulated it with images corrupted with impulsive noise. This program also used load balancing scheme, because of its heterogeneous computing environment. We will introduce basics of adaptive stack filter, and PVM first, and show suggested structure of new application program, and conclude with experimental results.

2. PARALLEL ADAPTIVE STACK FILTER

A stack filter is a sliding window nonlinear filter whose output at each window position is the result of a superposition of the outputs of a stack of positive Boolean functions operating on thresholded versions of the samples appearing in the filter's window.

Stack filters satisfy the two properties, which are the weak superposition property known as the threshold decomposition [3, 4], and the ordering property called the stacking property [3, 4, 5].

We can say a gray scale image X with pixel values ranging between 0 and M may be represented as the sum of series of binary-valued images,

$$X(s) = \sum_{l=1}^{M} x_l(s), \quad x_l(s) = \begin{cases} 1, & X(s) \geq l \\ 0, & X(s) < l \end{cases}$$

Now let W be the size P window of the filter and let $W(s)$ be the array of P points of the image that appears in the window W when its reference point is at position s. Then, the window array $W_X(s)$ of the image X can be similarly thresholded, so that

$$W_X(s) = \sum_{l=1}^{M} w_{X,l}(s)$$

Each stack filter $S_f(\cdot)$ is defined by a boolean function $f(\cdot)$ which satisfies a stacking property: if the output of f applied to $w_{X,l}(s)$ is 1, then the output produced when f is applied to threshold level k must also be a 1 if $k \leq l$. More formally, for all $k \leq l$

$$f(w_{X,k}(s)) \geq f(w_{X,l}(s))$$

A boolean function has this property if and only if it is positive. Due to these two properties, the operation of a stack filters is the same as the operation of the corresponding Boolean function for the thresholded binary inputs.

Optimal stack filter can be obtained by minimizing the mean absolute error between the output of the filter and some desired image [6]. If X is the desired image, and $\tilde{X}$ is the corrupted version observed by the filter, then the error to be minimized by proper choice of f is

$$\begin{aligned} MAE_f &= E\{|X(s) - S_f(W_{\tilde{X}}(s))|\} \\ &= E\{|\sum_{l=1}^{M}(x_l(s) - f(w_{\tilde{X},l}(s)))|\} \\ &\leq \sum_{l=1}^{M} E\{|x_l(s) - f(w_{\tilde{X},l}(s))|\} \qquad (1) \end{aligned}$$

By minimizing the bound in equation (1), we can find the Boolean function $f(\cdot)$ which makes the best decision at each location s as to whether the desired image value at s is less than l or not.

The optimal filtering problem can then be formulated as a zero-one integer linear program. However the number of constraints on f implied by the stacking property grows exponentially in the window size of the filter and knowledge of the joining statistics of the image X and the process which corrupted it are required for computing the coefficient of the cost function.

Adaptive stack filtering algorithm [7] were developed to minimize these problem, and new adaptive stack filtering algorithm was developed [1] to enhance the algorithm's parallel nature. Unlike the original algorithm, the stacking property will be enforced after L observation have been taken. If we suppose L as all possible observation from the training image and d_i as the i^{th} decision variable of boolean function $f(\cdot)$ for a boolean function $f(\cdot)$ can be completely specified by the decision vector $D = (d_1, d_2, \cdots, d_{2^P})$ where P is the size of window, then for each i, the number of increment or decrement on d_i is the cost incurred if the filter outputs a 0 or 1 for each observation respectively. Thus if d_i is thresholded at 0 to produce the boolean table, the result is the optimal hard decision. Though this is an optimal boolean function for filtering the corrupted image used in training process, it's not a positive boolean function until checking and enforcing the stacking property are applied. Two stacking property checking and enforcing schemes which can be parallelly implemented were also developed.

3. PARALLEL PROGRAMMING USING PVM

3.1. Heterogeneous Computing

We can define a heterogeneous computing environment as a group of computers which is composed of different architectures, fast network connecting all computers in the group, and programming environment familiar to users.

The heterogeneous computing environment can improve performance of the whole environment with relatively small costs, not being limited to a specific application. Unlike homogeneous environment, which will distribute given parallel functions evenly to its processors, in heterogeneous environment, programmers must analyze the features of parallel functions and consider the most appropriate mapping to a specific processor, because the loads to each host as well as their processing time will be varying. To accomplish this goal, rearranging an application parallelly, and examining the function code type of each rearranged function must be done, as well as mapping the code type to the benchmarking of each computer, and distributing the loads according to the result of mapping.

3.2. Parallel Virtual Machine(PVM)

PVM is a parallel programming environment which enables a programmer to treat heterogeneous programming environment as a parallel computer [2]. It uses the message passing methodology, which was most appropriate to heterogeneous environment and commonly used as a method of data communication of distributed computation.

PVM is composed of the user interface, which manages each local host, and libraries, which offer various message-passing functions enabling programmers to make parallel applications.

There can be many programming schemes using PVM, but the master-slave scheme is widely used. This scheme divides applications to two parts, main function part and parallel function part. The slave functions must be placed to each slave host compiled to be executed in each different architecture. This can be a constrained matter to programmers, but only the data which are required to perform the application need to be exchanged by message form, so the network loads are minimized.

4. FILTER STRUCTURE

The whole program is composed of two parts, training and filtering function, each of which use master-

slave scheme. The host used as the master provides user interface. It first reads the original and impulsive noise-corrupted version of images for training and filtering. Before getting into functions, the master performs benchmarking the current environment to distribute proper number of jobs to each host.

4.1. Master-slave structure

Master spawns slaves according to the PVM configuration information, which contains name of hosts currently enrolled in PVM environment, relative speed of each host, host architectures and task identification numbers, which is used internally to identify each host. Once master successfully spawned slave tasks to each host, master has the task id's which will be used to exchange messages between master and each task. Task id's have to be given to each slave, so that each slave can communicate with master or each other. Each task can send or receive messages with specific message type, so that each task can confirm that they got a message from right sender.

Functions for all of these works are provided in library, as well as the functions for each data type, which provide data packing procedure to prepare messages.

4.2. Load Balancing

In master-slave programming scheme, the whole process time is dependent on the time of the slowest slave, so we can reduce the whole process time by distributing proper number of tasks to each slave. In our program the number of tasks to be distributed is the intensity level of input image.

There are three important factor of benchmarking in heterogeneous computing environment:CPU speed, workload, and number of tasks to be given to each host. CPU speed will not be changed, once we measured it, but workload and number of tasks cannot be decided constantly. Workload of each host can be obtained by spawning each hosts a reasonable size of dummy task and measuring the elapsed time of the task. Because PVM communicate messages via network, the time needed to communicate via network is also important factor in our case. Network time also varies much according to the network loads of each time. Network time for communicating with each host can be obtained by measuring the time elapsed during sending a constant size of message to each host and receiving the message back.

Distributing tasks will increase the workload of each host, so we have to consider the workload after distribute tasks to each host. Let S_i be the CPU speed, W_i be the workload, N_i be the network load and d_i be the distribution of i^th host, then we can define a balanced distribution measure δ_i:

$$\delta_i = \frac{S_i}{W_i + N_i + d_i} \quad (0 \leq i \leq m-1, m = \text{number of hosts})$$

$$\sum_{i=0}^{m-1} d_i = N \quad (\text{where } N \text{ is the number of tasks})$$

Once W_i and N_i is obtained, we can balance δ_i by controlling number d_i of i^{th} host, so we can distribute proper number of tasks to each host.

4.3. Training Process

Two images, which are original image and noise-corrupted image, are sent to slaves to train stack filter. These images will be decomposed first and used to generate boolean function table. As mentioned above load balancing result is the number of binary image planes to be processed by each host.

Result of training function is a boolean function table, to which stacking property is not enforced. Although parallel algorithm of stacking property enforcing has been developed, the time needed to enforce stacking property in a serial manner outperformed parallel programming in PVM, because network load is greater than actual processing time.

The whole training function structure is depicted in figure 1.

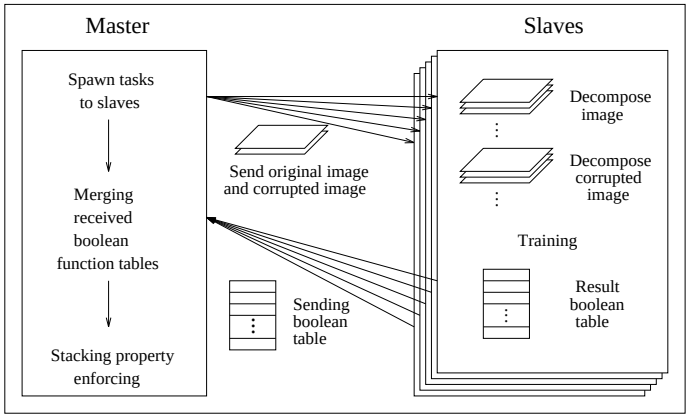


Figure 1: Training function structure

4.4. Filtering Process

Often size of boolean table is very large, though it is trained with relatively small size of window. The master sends the positive boolean function table to slaves only once when it spawns slave processes. It repeats

sending corrupted image and receiving filtered image until the error between input and output converges.

The whole filtering function structure is depicted in figure 2.

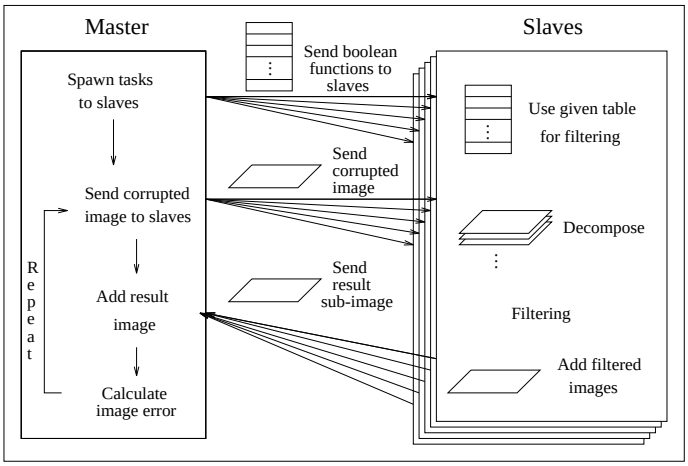


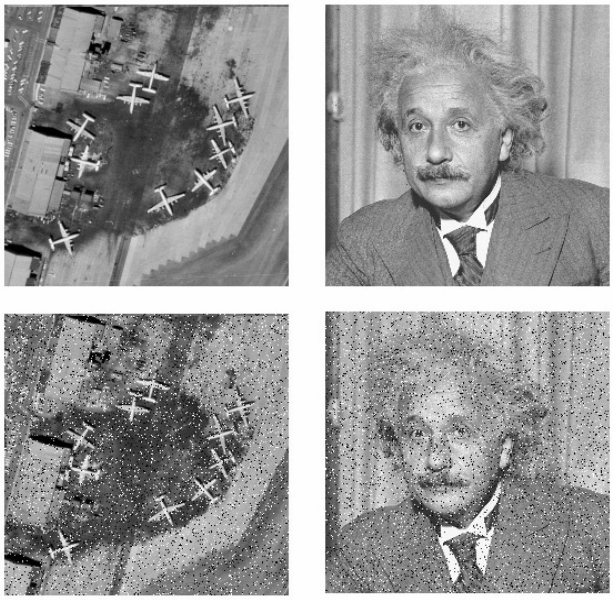
Figure 2: Filtering function structure

5. EXPERIMENTAL RESULT

The environment we used is composed of several different architectures, which are SUN sparc machines working on SUN OS, and Intel x86 machines working on LINUX. Table 1 shows the heterogeneous environment we used for experiments. We used photo "Aerial" (256×256,8 bits) and the photo "Albert" (512×512,8 bits) shown in figures 3(a) and 3(b) respectively, as noise-free original images. Stack filter was trained with images showed in figure 3(c) and 3(d) which are corrupted by impulses with 10% probability, each with 3×3 and 4×4 windows. The filtered results of corrupted images used in training are depicted in figure 4.

Table 2 shows the absolute error per pixel between each filtered output and the original noise-free image, and the execution time of the algorithm for the aerial photo and Albert, respectively. To compare the required network time and workloads of each hosts when the training process was performed, workloads and network time averaged by each process were also listed. Performance of this algorithm implemented on MasPar MP-1 parallel computer [1] was also showed for performance comparison.

Implementation on PVM showed better performance for the image size 512×512 with window of size 4×4, but not good for image size 256×256 and window of size 3×3. This result is due to the fact that the execution time on slave host is getting smaller as the image size and window size are getting smaller, although communication overheads will not be changed.



(a)	(b)
(c)	(d)

Figure 3: Images used in experiments;(a) Aerial photograph with 256×256 resolution, (b) Albert with 512×512 resolution, (c) *Aerial* , (d) *Albert*. Noisy images in (c) and (d) are corrupted by impulsive noise with an occurrence probability of 0.1.

6. CONCLUSION

In this paper, new implementation of parallel adaptive stack filter is developed using PVM. In order to maximize the merit of heterogeneous computing environment, a new scheduling algorithm which is based on the benchmarking appropriate to PVM computing environment was suggested.

The performance is dependent on the number of hosts used and the performance of each host, but reasonable number of hosts and performance is sufficient to show a good performance.

7. REFERENCES

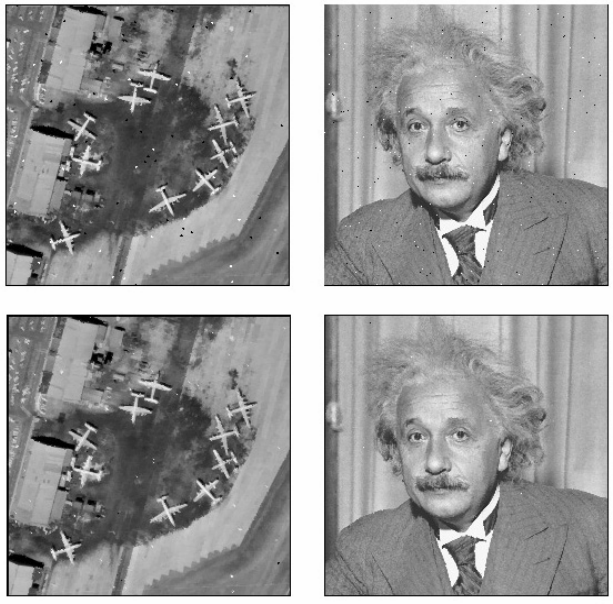
[1] J. Yoo, K.L. Fong, E.J. Coyle, G. B. Adams III "Fast algorithms for designing stack filters," 31'st Annual Allerton Conference on Communication, Control, and Computing, Monticello, IL Sep. 29 - Oct. 1 1993

Architecture	Number of machines	Relative CPU speed	Workload (average)	Network time (average)	Distribution
SUN Sparc 10	3	2000	2.14	0.30	30.0
SUN Ultra Sparc	3	4000	1.17	0.83	89.0
Intel Pentium 120MHz	2	2000	1.26	0.52	29.5
Intel Pentium 200MHz	2	6000	1.45	0.89	106.5

Table 1: PVM slave host environment used for experiments

Image	Window	Number of updates	Absolute Error	Training time (seconds)	Workloads (average)	Network time (average)	Error (MasPar)	Time (MasPar)
Aerial	3×3	10L	2.924	4.66	1.58	0.98	2.926	0.42
	4×4	20L	2.613	20.69	1.47	2.67	2.605	18.6
Albert	3×3	10L	3.223	18.16	1.59	4.36	3.223	0.83
	4×4	20L	2.705	40.12	1.50	5.02	2.705	70.4

Table 2: Performance measured with *Aerial* and *Albert* images



(a)	(b)
(c)	(d)

Figure 4: The filtered outputs by the application using PVM; (a) *Aerial* with 3×3 window, (b) *Albert* with 3×3 window, (c) *Aerial* with 4×4 window, (d) *Albert* with 4×4 window.

[2] Al Geist, A. Beguelin, Jack Dongarra, W. Jiang, R. Manchek, V. Sunderam "PVM : parallel virtual machine. A user's guide and tutorial for networked parallel computing," The MIT Press, 1994

[3] J.P. Fitch, E.J. Coyle, N.C. Gallagher, Jr. "Median filtering by threshold decomposition," IEEE Trans. on Acoustics, Speech, and Signal Processing, vol ASSP-32, no. 6, pp. 1183-1188, December 1984

[4] J.P. Fitch, E.J. Coyle, N.C. Gallagher, Jr. "Threshold decomposition of multidimensional ranked order operations," IEEE Trans. on Circuits and Systems, vol. CAS-32, no. 5, pp. 445-450, May 1985

[5] P.D. Wendt, E.J. Coyle, N.C. Gallagher, Jr. " Stack filters," IEEE Trans. on Acoustics, Speech, and Signal Processing, vol. ASSP-34, no. 4, pp. 898-911, August 1986

[6] E.J. Coyle, J-H. Lin, M. Gabbouj, "Optimal stack filtering and the estimation and structural approaches to image processing," IEEE Trans. on Acoustics, Speech, and Signal Processing, vol. 37, pp. 2037-2066, Dec. 1989

[7] J.-H. Lin, T.M. Sellke, E.J. Coyle, "Adaptive stack filtering under the mean absolute error criterion," IEEE Trans. on Acoustics, Speech, and Signal Processing, vol. 38, pp.938-954, June 1990