Performance Analysis of Median Filtering on Meiko^{TM} - a Distributed Multiprocessor System

K M Poon and N H C Yung

Department of Electrical & Electronic Engineering
The University of Hong Kong
Haking Wong Building, Pokfulam Road, HONG KONG
Tel: 852-2859-2685 Fax: 852-2559-8738 Email: nyung@hkueee.hku.hk

Abstract

This paper presents the performance analysis of realizing median filtering on a distributed multiprocessor system. The results of the performance analysis give a good indication of the performance gain in using multi-processor for median filtering over uni-processor. Such performance gain is proportional to the problem size as shown by varying the size of the image. Furthermore, through the analysis, it is clear that the computation time and inter-processor communications scale well with the number of processors in the system. However, the overall system performance does not have such behavior because of the initialization overhead dominating the computation time as the number of processors increases beyond a certain point. It is because of this relationship that an optimal performance is achievable with a certain number of processors. It is also found that this number varies with the problem size. In addition, the sub-image model is found to be an acceptable approach for this type of processing as only the necessary parts of the image are sent to the other processors. The master and slave scheme proves to be easy for programming, control and data manipulation. As a whole, this type of non-linear processing seems to fit well into the MIMD architecture.

1 Introduction

Typical digital images in practice are often degraded in some manner and to some extend that requires specially designed algorithmic steps to reduce or eliminate the degradation effect before they are displayed or further processed. The objective of image restoration is to determine an output image $\hat{f}(x,y)$ that resembles its original image f(x,y) as closely as possible [1, 2]. In order to achieve this objective, image restoration algorithms must be carefully chosen to deal with the type of degradation introduced at the input channels, transmission medium, sensor and/or digitizer. Typical degradation effects are blurring, distortion, additive random noise such as Gaussian white noise and salt-and-pepper noise; and signal-dependent noise such as speckle, film grain noise and quantization noise [3].

In general, image restoration algorithms are mathematically complex. Within this category, there is a subset of algorithms that are simple and heuristic, and has been widely employed to perform deblurring and noise removal in the spatial domain. Most of these algorithms are designed to deal with the more popular and practical types of signal-independent noise, e.g. Gaussian white noise, salt-and-pepper noise, burst channel errors and noise with a geometric structure [4].

Mathematically, an image degraded by additive random noise can be represented by

$$g(x,y) = f(x,y) + \eta(x,y) \qquad (1)$$

where g(x,y) is the degraded image of an original image f(x,y) due to an additive noise distribution $\eta(x,y)$. The process of noise removal may be interpreted as given g(x,y) and the knowledge of the statistical nature of $\eta(x,y)$, an approximation, $\hat{f}(x,y)$ of f(x,y) can be obtained. More specifically, $\hat{f}(x,y)$ may be calculated by

$$\hat{f}(x,y) = \underset{\substack{i=-N \\ j=-N}}{\overset{N \atop N}{T}} [g(x+i, y+j)] \qquad (2)$$

where T[.] denotes an (2N+1) by (2N+1) local window operator and N is an integer in this case.
For an image of size M by M, the number of pixels to be processed by each local window operation is M^2. If the local window operation is convolution-based, it

Meiko^{TM} is the trademark of Meiko Limited.

requires on average $(2N+1)^2$ multiplications and $(2N+1)^2-1$ additions to complete each local convolution. This gives a total of $2(2N+1)^2-1$ arithmetic operations per pixel by assuming that the speed in executing multiplication and addition are the same. If the local window operation is a ranking operation, it requires $(2N+1)\ln(2N+1)$ number of comparisons using a quick sort algorithm per pixel. This gives the following relationships

For convolution-based operator:
Number of x/+ per image = $M^2[2(2N+1)^2-1]$ (3)

For ranking-based operator:
Number of comparisons per image =
$M^2[(2N+1)\ln(2N+1)]$ (4)

From equations (3) and (4), the number of multiplications/additions and comparisons required by the two types of filtering operations are depicted in Table 1. It should be noted that a comparison can be assumed to be a lot slower than a multiplication or addition.

As can be seen in the above table, for typical M (512) and N (=2), the number of computations is comparable and rather large in both cases. If real-time response is desired, it is obvious that the processing platform should either be a very powerful and dedicated uniprocessor or a massively parallel multiprocessor. It is the intention of this paper to investigate the issues related to the realization of these computation intensive image processing algorithms on multiprocessors. The target platform chosen is the Meiko™, a distributed multiprocessor system and the target algorithm chosen for realization is the median filtering.

This paper presents the performance analysis of realizing median filtering on Meiko. The results of the performance analysis give a good indication of the performance gain in using multi-processor for median filtering over uni-processor. Such performance gain is proportional to the problem size as shown by varying the size of the image. Furthermore, through the analysis, it is clear that the computation time and inter-processor communications scale well with the number of processors in the system. However, the overall system performance does not have such behavior because of the initialization overhead dominating the computation time as the number of processors increases beyond a certain point. It is because of this relationship that an optimal performance is achievable with a certain number of processors. It is also found that this number varies with the problem size. In addition, the sub-image model is found to be an acceptable approach for this type of processing as only the necessary parts of the image are sent to the other processors. The master and slave scheme proves to be easy for programming, control and data manipulation. As a whole, this type of non-linear processing seems to fit well into the MIMD achitecture.

M	M^2	N	Window Size	Convolution-based	Ranking-based
		1	3 x 3	4,352	841
16	256	2	5 x 5	12,544	2,060
		3	7 x 7	24,832	3,485
		1	3 x 3	69,632	13,452
64	4096	2	5 x 5	200,704	32,952
		3	7 x 7	397,312	55,767
		1	3 x 3	278,528	53,821
128	16384	2	5 x 5	802,816	131,809
		3	7 x 7	1,589,248	223,068
		1	3 x 3	1,114,112	215,286
256	65536	2	5 x 5	3,211,264	527,237
		3	7 x 7	6,356,992	892,272
		1	3 x 3	4,456,448	861,143
512	262144	2	5 x 5	12,845,056	2,108,948
		3	7 x 7	25,427,968	3,569,091
		1	3 x 3	17,825,792	3,444,572
1024	1048576	2	5 x 5	51,380,224	8,435,794
		3	7 x 7	101,711,872	14,276,362

Table 1: Number of x/+ and comparisons for different N, M and Local Window Size

The paper is organized into a number of sections covering the aspects of basic median filtering, its merits and pitfalls; the Meiko™ computing surface; the implementation of median filtering in a distributed parallel environment and the analysis of the realization results. Discussions on the performance analysis and the lesson learnt will also be presented.

2 Median Filtering

Median filtering has been extensively used and is generally accepted as an effective algorithm for removing strong, spike-like components in a noisy image [5, 6, 7]. This non-linear filter algorithm is particularly suited for reducing salt-and-pepper noise with the advantage of preserving most of the edge information. As pointed out in [5], although noise suppression is obtained, signal distortion is introduced and some high spatial frequency features are lost. This manifests as a small degree of edge blurring in the restored image, which is not as large scale as those caused by some other filtering algorithms. This effect is not entirely undesirable if some of the fine details in the image are to be removed before object segmentation, or small gaps in lines or curves are to be filled. However, the distortion may be unacceptable visually and the reduction of the sharpness of lines, edges and boundaries may eventually affect the accuracy of the subsequent feature extraction and recognition processes.

Conceptually, median filtering is quite a simple algorithm. In the case of non-recursive median filtering, an odd number of pixels within a defined local window is ranked, and the median is taken as the filter output. Based on this concept, there exist many forms of median filter [7] such as the recursive median filter which has the filter output fed back to the filter and max/median filter which is reputed to remove noise as well as preserving geometrical features in the image. However, in our performance analysis, we will focus on a general non-recursive median filter which has the following definition.

For an image degraded by additive noise, we can assume equation (1) holds. In general, a two-dimensional (2N+1) by (2N+1) non-recursive median filter is defined by

$$\begin{aligned}\hat{f}(x,y) &= median\{g(x+i,y+j)| \\ i &= -N,..,-1,0,1,..N, j=-N,..-1,0,1,..N\}\end{aligned} \tag{5}$$

where $\hat{f}(x,y)$ is the restored image defined as the median of the pixel values enclosed in a two-dimensional window of size (2N+1) by (2N+1) centered at g(x,y).

An example is depicted in Figure 1 with N=2 and M≈ 500, a window size of 5×5 is used for determining $\hat{f}(x,y)$. The original image (Figure 1(a)) is a typical high-resolution head-and-shoulder picture of a lady. The noisy image (g(x,y)) is degraded by salt-and-pepper noise (Figure 1(b)) and Gaussian white noise (Figure 1(c)) of a signal-to-noise ratio (SNR) of -50 dB in each case. It can be seen from Figures 1(d) and 1(e) that the visual quality of the restored images is acceptable, although a number of features of this algorithm should be noted. Firstly, the line and edge information of the restored image is slightly distorted. This is manifested in the form of reduced sharpness in those features especially on the right hand background of the image. Secondly, the strongly clustered noise components in the Gaussian white noise case is not as effectively removed as the salt-and-pepper noise case. This is evident in the restored image (Figure 1(e)) where clustered noise components still remain although sparsely populated. This is due to the median determined by equation (5) is a noise pixel instead of a 'good' image pixel in those local regions with more noise pixels than 'good' pixels. Comparing this with other noise removing filters, median filter in fact performs much better than averaging, and is on a par with sigma filter in the Gaussian's case [8].

Comparative study of the computational complexity of the median filter and other filters such as the averaging filter, and sigma filter show that median filtering requires more computational resources than the averaging filter, max/min filters and even the sigma filter because of the comparison operation and the number of comparisons needed in sorting [8]. This becomes worst when N increases. Typical value of N is two or three. For N equals 2, measured timing statistic shows that the median filter is roughly 5.25 times slower than an average filter and 1.2 times slower than a sigma filter.

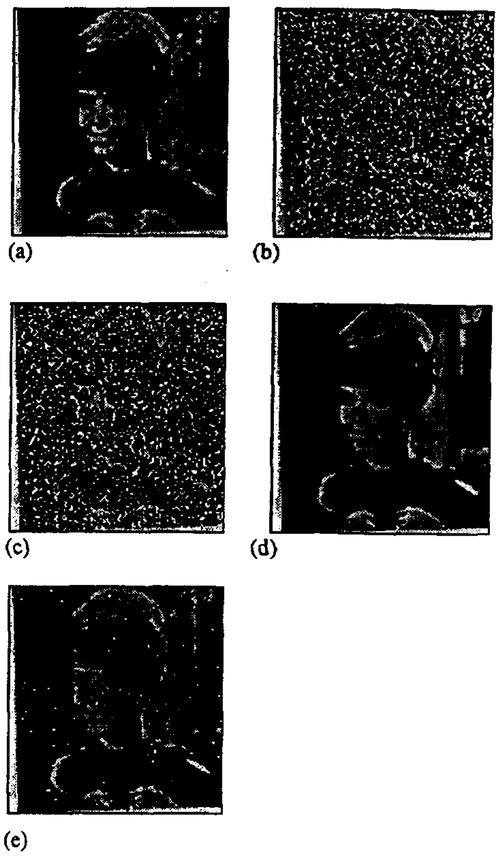

Figure 1: Median filtering of a 477 × 505 image using a 5x5 window size (a) Original image; (b) Noisy image degraded by salt-and-pepper noise; (c) Noisy image degraded by Gaussian noise; (d) Restored image of (b); (e) Restored image of (c).

In order to make median filter a viable algorithm for practical noise removal in terms of both noise elimination capability and computational speed, parallel processing is an attractive proposition. Although certain speed-up factor is anticipated when parallel processing systems are employed for the filter realization, understanding must be obtained with regard to the optimum number of processors used, communication overhead between processors and the overall delay including the input and output of the large image array. Therefore, it is the intention of the following sections to explore these areas, and especially in

Section 4, determine whether there exist an optimum processing configuration, how the communication scheme between processors affect the overall performance, and the effect of moving a large image through the input and output channels.

3 Meiko™ Computing Surface

The Meiko™ Computing Surface is a RISC-based distributed-memory multiprocessor system. It consists of three types of processors in the whole system: the INTEL i860, the SUN SPARC 2 and INMOS T800 transputers. The INTEL i860 processors are the crux of the whole system where all the parallel computations are performed. In our current configuration, there are 16 i860 processors in the Meiko™ Computing Surface altogether. The SUN SPARC 2 acts as the host to the i860 processors whereas the INMOS T800 transputers are solely for communications and interconnections between the i860 processors. The Meiko™ system provides a software configurable multi-stage crossbar communication network (Computing Surface Network, CSN) as the basis for exchanging data between different processing elements. In other words, a processing element can be connected directly to any processing elements in the system, subjected to comparable delays. The system block diagram is depicted in figure 2. Theoretically, there is no upper bound on the number of i860s to be installed on a Meiko™ Computing Surface. However, there should be a trade-off between the speed up, communication efficiency and the cost of the whole system.

Physically, two i860 processors are accommodated on a single MK096 board. Each board also contains four T800 transputers for communication with the CSN network. Each i860 processor is supported by 16 to 32 MB local memory and eight transputer links giving a total bandwidth of 160 Mbps per processor. At present, Meiko™ has twelve i860 with 16 MB memory and four with 32 MB. This gives a total of 320MB memory for parallel programming.

The MK083 board has a SPARC 2 RISC processor at 40 MHz with 64 MB of memory and 64 KB cache. It is used to boot strap the i860 processors and perform the resource management of the CSN. The SPARC host running the SunOS 4.1.3 provides an UNIX environment with C and FORTRAN compilers for the i860 processors. It also controls all the i860 processing elements and provides I/O operations for the system.

The Meiko™ Computing Surface is classified as a distributed-memory MIMD (Multiple Instruction streams Multiple Data streams) system according to Flynn's taxonomy. An MIMD model is the most general and powerful as compared with other parallel architectural models like SIMD and SISD. There are many examples of MIMD machines, like Convex C3, CRI Cray-3, CRI Cray Y-MP, CRI T3D, IBM 9076 SP1, IBM SP2, Meiko CS, Meiko CS2, nCube 2 and TM CM-5. For MIMD machines, each processor can execute different codes on different data on different sub-problems of a single problem. Therefore, the communication between processing elements is an extremely important issue in such machine.

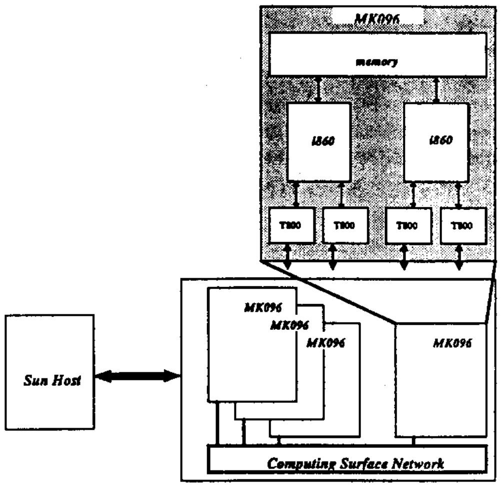

Figure 2: Block diagram of Meiko™ Computing Surface

The Meiko™ Computing Surface also has fully supported parallel processing library of routines, programming language, debugging tools and visualization software. They provide a certain degree of ease in parallel programming within an MIMD environment. It is such programming support that motives the implementation and performance analysis of median filtering, of which the problem can be subdivided and executed in individual processing elements nicely.

4. Performance Analysis

4.1 Implementation of Median Filtering - a Master/Slave Approach

In our evaluation, the degraded image of pixel size 477x505 is initially stored in the Sun SPARC host. The host is responsible for all the operating system operations, initialization and downloading of programs to the master and slave processors. An i860 processor is assigned as the *master* and it is responsible for the reading and writing of the image data from/to the secondary storage device attached to the host, as well as dispatching the sub-images to the slave processors. In addition, the master itself also takes part in the filtering of a sub-image. Once the image is downloaded into the master, it is then divided into horizontal sub-images determined by the number of slave processors involved in the processing as given in equation (6). The 2N term in each bracket takes into account the edge condition spatially, where N has the usual definition. This can be seen more clearly from figure 3. This arrangement adds extra communication for the master, however it avoids the message passing communication between the processors during the filtering process and hence reduces the chance of blocking due to communication. Due to this consideration, when the sub-images are processed by the slaves and returned to the master, a simple reconstruction of the image is required as each filtered sub-image will have the size of $[\text{image width}]\times\left[\frac{\text{image height}}{\text{number of slaves } +1}\right]$. Each sub-image is then sent to a slave i860 processor. The role of the slave processor is simply to perform the median filtering on the sub-image. Sub-image size =

$$[\text{image width}+2\text{N}]\times\left[\frac{\text{image height}}{\text{number of slaves } +1}+2N\right] \quad (6)$$

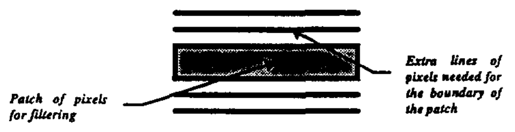
Patch of pixels for filtering
Extra lines of pixels needed for the boundary of the patch

Figure 3: Pixels transmitted to the slaves

Due to the inherent spatial locality of the median filtering process, the only communication between processors is at the beginning and the end of the median filtering process. In essence, the master processor dispatches the divided sub-images to the slave processors and collates the filtered sub-images at the end of the process through the message passing routines provided by Meiko™. This parallelization of the filtering process is depicted in figure 4.

The advantage of using a master/slave approach is that the read/write operations and the interfacing with the host are managed solely by the master processor and the slave processors are left to carry out the filtering process. Furthermore, the slave processors are only given the part of the image they need to process instead of the whole image. As a result, the amount of unnecessary communications is cut down.

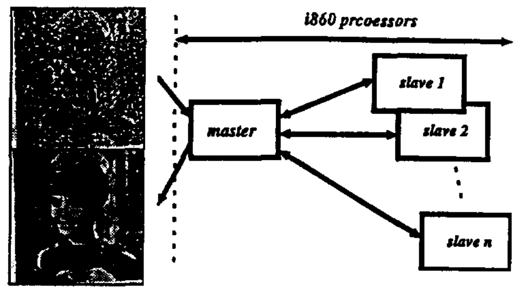

Figure 4: Visualization of parallelizing the filtering process

The measurement of the system performance is taken from the master's *user and system time*. This master's user and system time is the time that the master needed to dispatch the sub-images to the slaves, filter its own sub-image and collect the sub-images from the slaves to reconstruct the final filtered image (equation 7). It does not account for reading and writing the image from and to the secondary storage device of the host. In addition to the user and system time, the *turnaround time* of each complete run is also measured. The turnaround time is the elapsed time for a complete run of the program. This accounts for the operating system overhead, the bootstraping of the processors, the loading of the executable codes into each processor, copying of the image from host disk to the master, t_{master} and the writing of the filtered image to the host disk. This parameter is given by equation (8)

$$t_{master} = t_{dispatch} + t_{median} + t_{receive} \quad (7)$$

where

t_{master}=	master's user and system time
$t_{dispatch}$=	time for dispatching all the sub-images to the slaves
t_{median}=	time for performing a median filtering on a sub-image
$t_{receive}$ =	time for collating all the sub-images from the slaves to construct the filtered image.

$$t_{turnaround} = t_{os} + t_{p-init} + t_{write} + t_{master} + t_{read} \quad (8)$$

where

$t_{turnaround}$=	turnaround time for each complete median filtering
t_{os}=	operating system overhead
$t_{p\text{-}init}$=	time for processor initialization
t_{write}=	time for writing the whole image into the master
t_{read} =	time for reading the whole image from the master

4.2 Results

Figures 5-9 summarize the results obtained from implementing the above configuration, and processing different image sizes on Meiko™. The number of i860 processors implemented ranges from 1 to 12 for each case. All the images concerned are 8-bit gray-scale head-and-shoulder image of a lady having the same signal-to-noise ratio of -50dB degraded by Gaussian white noise.

Figure 5 depicts the performance of median filtering on a 477×505 image. It plots the two time parameters (t_{master} and $t_{turnaround}$) in seconds versus the number of processors used. As can be seen in the figure, a single i860 takes about 47 seconds to complete the median filtering process. For reference purpose, an identical median filtering process running on a 486 DX2/66 takes around 84 seconds to complete. The mean-square errors of the filtered images are the same in both cases. As the number of processors increases, t_{master} decreases steadily, which can be approximately roughly by equation (9)

$$t_{master}(n) = \frac{t_{master}(1)}{n}, \qquad (9)$$

where n (n≥1) is the number of slave processors. For example, for n=11, $t_{master}(11) = t_{master}(1)/11 \approx 2.8$ seconds. If t_{master} is our only consideration, then computational speed-up will be proportional to the number of processors in the distributed system. In reality, this is seldom the case as the reading and writing of the image data from and to the secondary storage device play an important role in the timing equation, as well as the overhead required by the operating system and the overhead incurred in the processor initialization. Therefore, $t_{turnaround}$ is a more realistic parameter to be considered here.

When considering $t_{turnaround}$, its behavior exhibits different characteristic from t_{master}. As from figure 5, $t_{turnaround}$ appears to reach a minimum at 5 or 6 processors, and beyond that, the improvement is minimal. For a single processor configuration, $t_{turnaround}$ is about 47 seconds, which is 16 second more than t_{master}. For a 12 processor configuration, the difference is about 20 second instead, which represents a steady increase as the number of processors in the system is increased. This difference represents the sum of t_{os}, $t_{p\text{-}init}$, t_{write} and t_{read}. Out of this list of timing parameters, t_{os}, t_{write} and t_{read} should be relatively constant, independent of the number of processors used in the distributed system. However, $t_{p\text{-}init}$ is not so as it is determined by the time required to bootstrap all the processors in the system, and the time required to load the individual executable code segment into each processor. For processor bootstraping, the time required is obviously proportional to the number of processors used.

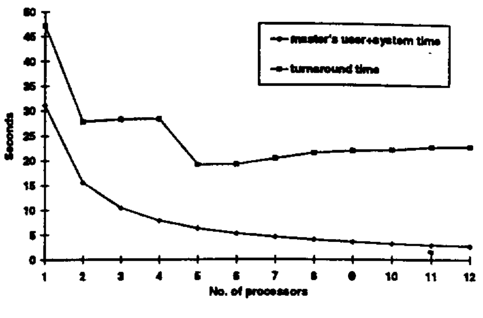

Figure 5: Performance of median filtering on 477×505 image degraded by Gaussian noise

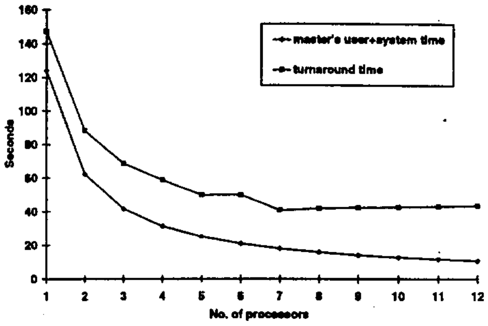

Figure 6: Performance of median filtering on 954×1010 pixel image degraded by Gaussian noise

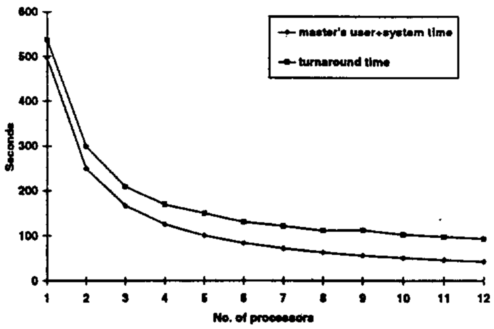

Figure 7: Performance of median filtering on 1908×2020 pixel image degraded by Gaussian noise

Similarly, the executable code loading time also bears such relationship with the number of processors. With this overhead mainly determined by the number of processors used in the system, $t_{turnaround}$ as a result is also determined by the number of processors in the system.

When the number of processors is small, t_{master} is a dominating factor in $t_{turnaround}$, and therefore, increasing the processor number will improve the overall performance. However, when the number of processors is large, the relationship between $t_{p\text{-}init}$ becomes the dominating factor in $t_{turnaround}$, and therefore, increasing the number of processors will degrade the overall performance. Although this rise in $t_{turnaround}$ beyond the minimum number is rather gradual (about 5 seconds altogether), the indication is that the optimum number of processors should be employed in solving this type and size of problem is 5 or 6.

Increase the size of image to 954×1010 pixels, which is 4 times the image in figure 1, the timing result is shown in figure 6. It depicts a similar phenomenon as in figure 5. The t_{master} time shows a exponential decay. It should also be noted that the corresponding t_{master} in figure 6 is exactly 4 times of that in figure 5. Such speed up of the median filtering is nearly ideal if we consider scaling the problem by size or by the number of processors used. On the other hand, it reaches a minimum at 7 processors for the $t_{turnaround}$ curve. Although the subsequent rise in turnaround time is small, there is no further sign of decline in $t_{turnaround}$ after 7 processors. This is believed to be due to the same reason as in the case of smaller image size. It should also be noted that $t_{turnaround}$ is less that 4 times of the corresponding $t_{turnaround}$ in figure 5. This can be explained that the time for the operating system, reading and writing an image from and to the master is not increasing in the same proportion as t_{master}. However, it should be noted that the difference between $t_{turnaround}$ and t_{master} becomes larger when more processors are added to the system. This can be explained as $t_{p\text{-}init}$ becomes a lot more prominent compared with the computational activities in the processors. For example, in a single processor configuration, t_{master} is 125 seconds, $t_{turnaround}$ is 147 seconds, giving a difference of 22 seconds, 14.9% of $t_{turnaround}$. In the case of the 12 processor configuration, t_{master} is 12 seconds, $t_{turnaround}$ is 43 seconds, giving a difference of 31 seconds, 75.6% of $t_{turnaround}$.

Figure 7 is the timing result of filtering the same image of size 1908×2020 pixels. Again the t_{master} curve behaves in an expected way. The parameter $t_{turnaround}$ declines all the way from 1 to 12 processors and the value is less then the corresponding $t_{turnaround}$ in figures 5 and figure 6 by 4 and 16 times respectively. However, the difference between the t_{master} curve and the $t_{turnaround}$ curve increases as the number of processors increases. In this case, the difference varies from around 9% of $t_{turnaround}$ to about 50% of $t_{turnaround}$, which again, indicates the dominating factor in the system changes from the computation to initialization when the number of processors in the system is increased from 1 to 12.

Figure 8 and figure 9 depict the combined results from figures 5-7 with logarithmic y-axis (time axis). In figure 8, the logarithmic y-axis reflects the growth of the image size with respect to the increase in $t_{turnaround}$. The differences between the three curves is the multiplicating factor of the $t_{turnaround}$ on increasing the image size. The differences between the 1× and 4× curves is smaller than that of 4× and 16× curves.

Due to the communication overhead of distributing and collecting the sub-images, increasing the image size by *n* times does not imply $t_{turnaround}$ would also be increased by *n* times. Of course, the communication overhead increases with the images size, so is the computations. The determining factor is which parameter grows faster. If the communication overhead grows faster, then much of the time is spent on communication rather than the useful filtering operations. On the other hand, if the reverse is true, the system spends more of its time on filtering and less on communication. This implies a more efficient operation as a whole.

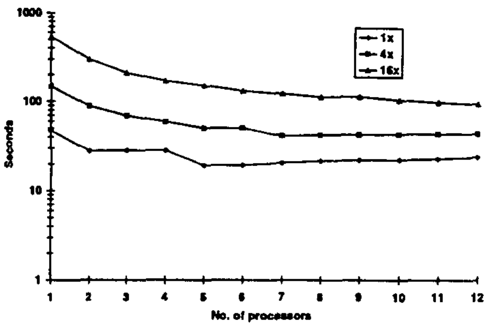

Figure 8: Turnaround time of median filtering on different size of image

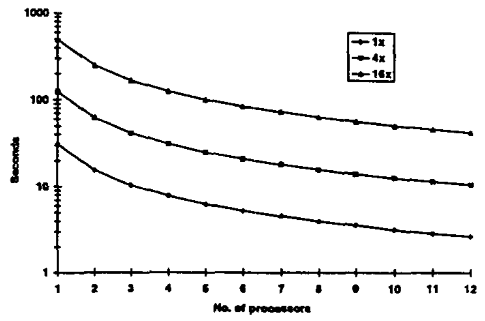

Figure 9: Master's user and system time on median filtering different size of image

Figure 9 is a comparison of t_{master} with different sizes of images. The time differences between the three curves are fairly constant. This graph indicates that t is both scalable in the direction of increasing the problem size and the parallelism of more processing elements.

5 Discussions

The t_{master} parameter reflects the actual computational load of the median filtering on the multi-processor system. It is scalable with respect to the problem size of the image. This is due to the individuality of computation on each pixel, which requires no communication during the filtering process. Thus, t_{master} scales well on the problem size.

However, $t_{turnaround}$ is not scalable. Increasing the image size by *n* times does not imply $t_{turnaround}$ would increase by the same proportion. When the problem size is small, $t_{turnaround}$ would increase as the number of processors used is more than 5 (figure 5). Such trend of not being scalable is also shown in figure 6 with more than 7 processors. This behavior can be accounted for by the overhead required by the processor bootstraping and by loading the execution codes onto the processors. Disregarding the number of processors used in the system, the other overheads due to the operating system, reading and writing of the image from and to the memory are relatively constant for a fixed size image. Although the increase in tturnaround is not significant, it can be deduced that the overall performance will not improve much when the number of processors in the system is more than a certain number. In addition, the optimal number of processors is dependent on the problem size. For an 477×505 image, the optimum is 5 or 6 processors, and it is 7 for an 954×1010 image. In these cases, using more processing element does not imply a gain in the overall performance.

In our computational model, the image is distributed as sub-images to the slave processors, this is exactly where the overhead of the parallel computation is. In principle, there are two possible extreme approaches for communicating the image data between the processors. The first one is that the master distributes a copy of the image to each of the slave processor. Upon receiving the image copy, the slave processor calculates the part that it is responsible for. When the calculation is completed, the image copy is sent back to the master. The master collates all the copies into a final filtered image. This approach saves the computation of subdividing the image into sub-images by the master. However, communication overhead increases, which could be significant when the image is large. Although the communication bandwidth allows such overhead (for small image size), it is undesirable to send a large amount of irrelevant pixels to the slave processors. The second approach is to pass pixel by pixel to the slave processors. The master is only responsible for the communications of sending and receiving pixels. In such case, the master stays idle for most of the time waiting for synchronization. Of course, both methods mentioned above are far from satisfactory. Our approach in sending a sub-image to a slave processor makes the necessary and reasonable compromise between these two extreme cases.

The MIMD architecture with message passing interface between processing elements performs well on median filtering. If share memory is implemented, the bottleneck will be in the communication channel through which pixels are accessed. The effect is more serious for larger number of processing elements in the system. On the other hand, the SIMD architecture is only suitable for linear pattern of code execution and computational job in which each pixel is subjected to similar processing. In median filtering, the major operation is on sorting a group of integer numbers. This is a non-linear process as the number of swapping processes varies from pixel to pixel. In addition, the bottleneck on the share memory still exists.

If we consider the overhead due to parallelizing the median filtering algorithm, this overhead can be calculated by equation (10) and depicted in figure 10.

$$\text{Overhead} = \frac{t_{turnaround} - t_{master}}{t_{turnaround}} \qquad (10)$$

The overall trend is that the overhead percentage increases as the number of processors increases. This indicates the scenario of when the processing system consists of a large number of processors, much of its time during execution is not spent on the actual filtering computation, but on initialization and communications. The overhead percentage decreases as the image size increases. This corresponds to the fact that more time is now spent on processing the image in comparison with the overhead time. In other words, the growth rate of the initialization/communication overhead is less than the growth rate of the computational requirement due to the increase in problem size.

Finally, parallelizing image filtering involves the partitioning of an image into sub-images and dispatching the individual sub-images to the slave processors. The granularity of parallelism is thus very coarse. This is also true for filters based on calculating the convolution between a local neighborhood and a window of coefficients. Unless we have a good optimizing parallel compiler, to achieve fine granularity is a uphill task for parallel language programmer.

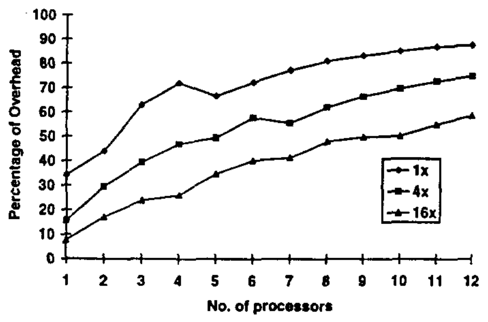

Figure 10: Percentage of overhead on median filtering

6 Conclusion

Firstly, it is evident that there is a definite gain in performance when computation intensive task such as median filtering is realized in a multi-processor system instead of a uni-processor system. As can be seen in figure 5, for $M \approx 500$ and $N=2$, the largest performance gain is about 2.5 times between 5 processors and 1 processor. Further analysis results shown in figures 6 and 7 indicate the performance gain increases as M increases. For $M \approx 1000$, the gain is about 3.7 times whereas for $M \approx 1500$, the gain is about 5.5 times. This also indicates that the gain in performance is proportional to the size of the problem.

Secondly, increasing the number of processors in the system does not always imply an increase in overall performance. This is due to the fact that the parallelizing overhead (t_{os}, $t_{p\text{-}init}$ and etc.) increases with the number of processors, even though t_{master}, the computational and processor communication time decreases with the number of processors. It is because of this relationship that the overall performance curve shows an optimal point corresponding to a certain number of processors. For less than this number, t_{master} dominates, and for more than this number, the parallelizing overhead time dominates.

Thirdly, the sub-image model is found to be an acceptable approach for this type of processing as only the necessary parts of the image are sent to the other processors. The master and slave scheme proves to be easy for programming, control and data manipulation. In addition, this type of non-linear processing seems to fit well onto the MIMD architecture.

Finally, this coarse grain parallelization points to the potential of speeding up computation intensive tasks such as the median filtering shown in this paper. This potential may be further exploited if fine grain parallelization becomes more readily available. This requires good optimizing parallel compilers and perhaps other parallelizing tools to achieve the goal.

7 References

[1] R. C. Gonzalez & R. E. Woods, "Digital Image Processing", Addison-Wesley, 1992.

[2] J. S. Lim, "Two-Dimensional Signal & Image Processing", Prentice-Hall, 1990, pp.524-588.

[3] N. D. Sidiropoulos, J. S. Baras & C A Berenstein, "Optimal filtering of digital binary images corrupted by union/intersection noise", *IEEE Trans. on Image Processing,* Vol.3, No.4, July 1994, pp.382-403.

[4] J. S. Lee, "The sigma filter and its application to speckle smoothing of synthetic aperture radar image", in *Statistical Signal Processing* edited by I. W. Edward, G. S. Smith, Marcel Dekker, New York, 1984, pp.445-459.

[5] G. R. Arce & M. P. McLoughlin, "Theoretical analysis of the Max/Median Filter", *IEEE Trans. on ASSP,* Vol.ASSP-35, No.1, Jan 1987, pp.60-69.

[6] Zheug Yi, "Image analysis, modeling, enhancement, restoration, feature extraction and their application in nondestructive evaluation and radio astronomy", *Ph.D. Thesis,* Dept of EE, Iowa State University, USA, 1987.

[7] J. E. Hall & J. D. Awtrey, "Real-time image enhancement using 3x3 pixel neighbourhood operation functions", in *Selected Papers on Digital Image Processing,* Vol. MS17, Ed. by Mohan M. Trivedi, SPIE Optical Engineering Press, 1990, pp.595-598.

[8] N H C Yung & H S Lai, "An intelligent spatial filtering algorithm for removing additive random noise in digital images", *Research Report,* Department of Electrical & Electronic Engineering, The University of Hong Kong, 1994, pp.23-59.

[9] L. A. Crowl, "How to measure, present, and compare parallel performance", *IEEE Parallel & Distributed Technology, Systems & Application,* Vol.2, No.1, Spring 1994, pp.9-25.