Journal of Engineering and Applied Sciences 12 (12): 3083-3089, 2017
ISSN: 1816-949X
© Medwell Journals, 2017

# Implementation of Various Edge Detection Filters Using Different Thread Distributions

Hezil Renita Dsouza, Jayashree Ballal and S. Pooja
Department of Information and Communication Technology,
Manipal Institute of Technology, 576104 Manipal, India

**Abstract:** A dedicated framework with memory interleaving and parallel handling strategies can lessen the weight of host CPU along these lines making the framework more appropriate for ongoing applications. Presently it is conceivable to use parallelism utilizing multi-cores on CPU however it should be utilized explicitly to gain superior performance. Latest GPUs has a generous amount of cores and it has a capacity for superior performance in generally valuable applications. Graphical Processing Units (GPUs) have turned out to be imperative in giving handling power to superior performance applications. CUDA is a programming interface for GPU processing and it is an exclusive programming interface and collection of language extensions which works just on NVIDIA's GPUs. In this study, some of the image processing methods namely, Sobel, Prewitt and Robert's Cross edge detection are introduced and executed using different thread distributions and compared with the sequential implementation, i.e., single core CPU and multiple-core CPU. Execution outcomes show that critical speedup is accomplished with the usage of GPU as compared to single-core CPU and multiple-core CPU. It is also observed that the speedup increases with the increase in image size.

**Key words:** CPU, GPU, single-core, multiple-core, edge detection

## INTRODUCTION

In the present era of technology, high performance computing is playing very vast role. Over the past several decades, various parallel programming languages and models have been developed which are mainly based on two different hardware technologies, namely, the Central Processing Units (CPUs) and the Graphics Processing Units (GPUs). The main goal of developing advanced computer architectures and usage of existing present day frameworks is to run greater and more convoluted applications speedier after some time. The need for extended figuring power drove in the late 1980's to change the high parallel adaptable multiprocessing structures.

Generally, programming has been created for serial computations. Here, the problem is divided into separate pieces and they are executed progressively in an enduring movement on a single processor. Just a single instruction may keep running at a solitary snapshot of time. Parallel processing is the concurrent usage of various PCs to deal with a computational issue. The problem is divided into separate pieces that can be clarified at the same time. Each part is additionally isolated to a forward movement and executed in the meanwhile on different processors.

**General scenario:** Customarily, applications have been developed for serial computation which keeps running on a solitary PC having a solitary Central Processing Unit (CPU) centre. Here the problem is divided into separate pieces that can be solved simultaneously. That is just a single instruction may execute at any moment in time. Advantage of this is a larger number of variables can be held in registers simultaneously by improving the performance of code that accesses those variables. This makes the debugging process easier. Concurrent computing is a type of processing in which a few calculations are executing during overlapping time periods simultaneously rather than successively (one finishing before the following begins), i.e., concurrent computing is one where a computation can achieve progress without sitting tight for every single other computation to finish where more than one calculation can progress in the same time.

**Why parallelism:** To improve the speed of execution of a program, the frequency of the processor may be increased. But as consequence of increased frequency the processor may consume more power as shown by the following Eq. 1:

$$P = \frac{1}{2}CV^2f \quad (1)$$

**Corresponding Author:** Hezil Renita Dsouza, Department of Information and Communication Technology, Manipal Institute of Technology, 576104 Manipal, India

Where:

P = Power consumed by the processor
C = The capacitance offered by the wires
V = The switching voltage of the transistor
f = The frequency

Due to the increased power consumption excessive heat will be generated which may damage the processor. As given by the Moores law, component density doubles every 18 months. As a result of increased component density, the modern processors are having more than one processing cores which operate at moderate speed. Most of the applications are inherently parallel in nature. Therefore job can be split up into multiple subtasks and executed simultaneously on multiple cores. This improves the overall speed of the program execution.

**OpenMP:** OpenMP is a usage of multiple threads; it's a technique for parallelizing whereby an main thread splits into a predefined number of sub threads and the structure separates an assignment among them. These sub-threads then run simultaneously with the runtime condition designating threads to various processors.

The fragment of code that is planned to keep running in parallel will be identified with an pre-processor directive order that will make the threads frame before the segment is executed. Each thread consists of an unique id connected to it. This id can be obtained utilizing a function called omp_get_thread_num. The thread id is a numeric number and the main thread has unique id of 0. Once a parallelized code get executed, the threads join over into the main thread which proceeds ahead to the finish off the program. As a matter of course, each thread executes the part of parallelized code autonomously. To ensure that each thread executes its designated portion of code, task among the threads is partitioned using work builds.

**GPU parallel computing:** GPU computing is utilization of an Graphics Processing Unit (GPU) together with a CPU to quicken logical, engineering and venture applications. A Graphical Processing Unit (GPU) is a specific electronic circuit intended to quickly control and adjust memory to quicken the production of pictures in a frame buffer planned for output to a display.

GPUs are extremely productive at controlling computer graphics and image processing also their significantly parallel structure makes them more powerful than all around helpful CPUs for calculations where the huge blocks of visual data are processed in parallel.

**Literature review:** There have been several researches aiming at implementing the various applications on CPU and GPU. GPU offers the execution of several threads in parallel and provide higher throughput as compared to the traditional multi-core architecture.

Arnautovi *et al*. stated that the effective vehicle courses are an important logistics issue which has been contemplated for quite a few years. Metaheuristic algorithms offer a few answers for that issue. It manages successful execution of the Ant Colony Optimization algorithm (ACO) (Arnautovic *et al*., 2013) utilizing GPU, it can be utilized to locate the better vehicle course between source and destination. The method is concerned on searching the shortest way in several possible ways. It is embarrassingly parallel, since every ant develops a conceivable issue arrangement autonomously. Consequences of consecutive and parallelized usage of the calculation are introduced. An examination concentrated on executing ACO utilizing OpenMP and CUDA gives a premise to investigation of various outcomes accomplished on those two stages (Dawson and Stewart, 2014).

Karimi *et al*. (2010) demonstrated the execution of CUDA and OpenCL utilizing perplexing, close indistinguishable kernels (Bhardwaj and Mittal, 2012). The execution tests measure and contrast information exchange times with and from the GPU, kernel execution times and end-to-end application execution times for both CUDA and OpenCL. CUDA performed better while exchanging information to and from the GPU. There were no impressive changes in OpenCL's relative information exchange execution as more information was exchanged. CUDA's kernel execution was likewise reliably speedier than OpenCL's, regardless of the two usages running almost indistinguishable code. CUDA is by all accounts a superior decision for applications where accomplishing high execution speed is necessary. Generally, the decision amongst CUDA and OpenCL can be made by considering elements, for example, earlier nature with either framework or accessible improvement devices for the target GPU equipment.

Wang *et al*. implemented parallel sorting algorithms on GPU and presented an analysis of parallel also sequential bitonic, odd-even and rank-sort algorithms on different GPU and CPU architectures. The execution for different queue sizes is measured concerning sorting time and rate and furthermore the speedup of bitonic sort over odd-even sorting algorithms is appeared on different GPUs and CPU (Burkitt *et al*., 2010). The result demonstrated that 19x accelerate of bitonic sort against odd-even sorting method for smaller queue sizes on CPU and maximum of 2300x speedup for large queue sizes.

Bitonic sort performs well for exceptionally bigger line sizes on both Nvidia Quadro 6000 and GTX 260. Scene recognition (Chouchene *et al*., 2014) has transformed into a bewildering field in picture processing zone and various procedures have been proposed starting late in which expelling the scene significance from overall components has been exhibited to have higher higher retrieval efficiency exactness (Lu *et al*., 2012). However, the procedure of essence is overwhelmingly dull and not proper for consistent application. Here, the CUDA designing is passed on to stimulate this methodology.

Tests speak to obvious speedup differentiated and usage using just CPU. This gives speedier and more viable multithreaded execution on both broadly useful realistic handling units and multi-core CPU. Here CUDA is passed on for significance extraction and OpenMP is associated for continuous similitude correlation stage and shows speedier computational conditions on the proposed building than on just CPU.

In Natural Language Processing (NLP) applications (Gupta and Babu, 2011), the key process is string sorting out in perspective of the huge size of vocabulary. In string arranging strategies, information reliance is unimportant and in this way it is perfect for parallelization. It is conceivable to apply parallelism utilizing multi-focuses on CPU yet they should be utilized explicitly to complete predominantly. Rajasekhara *et al*. investigated the execution of single-core, multiple-core CPU and GPU utilizing a natural language processing application (Gupta and Babu, 2011). Outcomes demonstrated that multi-core CPU has favoured execution over the single-core CPU; however a GPU framework has unmistakably beaten them with much better execution over CPU for Natural Language Processing (NLP) applications.

In the most of the above approaches, GPU offer the possibility of executing several threads in parallel, providing the user with higher throughput with respect to traditional multi-core processors.

## MATERIALS AND METHODS

**Basics of edge detection algorithm:** This study lists the various edge detection filters considered for this project and briefs about it.

**Sobel filter:** Sobel edge detection takes two 3*3 kernels into consideration which then are combined with the source image to calculate the derivative approximations (Khan *et al*., 2011). One kernel is used for horizontal changes while the other stands for vertical changes. If A is characterized as input image and if $G_x$ consists the horizontal derivative and $G_y$ consists the vertical derivative, the calculations are stated as follows:

$$G_x = \begin{matrix} +1 & 0 & -1 \\ +1 & 0 & -2 \\ +1 & 0 & -1 \end{matrix} * A \qquad G_y = \begin{matrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix} * A$$

X-coordinate can be characterized as expanding towards "right"-direction; y-coordinate can be characterized as expanding towards "down"-direction. With respect to every point in the picture, subsequent slope calculations could be consolidated to form the gradient magnitude, using:

$$\theta = a\tan\left(\frac{G_y}{G_x}\right) \tag{2}$$

Using the above info the gradient direction is calculated as follows:

$$G = \sqrt{G_x^2 + G_y^2} \tag{3}$$

**Prewitt filter:** Prewitt operator takes into consideration two 3*3 kernels that are combined with the input picture to compute calculations of the derivatives (Wang *et al*., 2009). One kernel is used for horizontal changes while the other stands for vertical changes. If A is characterized as input image and if $G_x$ consists the horizontal derivative and $G_y$ consists the vertical derivative, the calculations are stated as follows:

$$G_x = \begin{matrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{matrix} * A \qquad G_y = \begin{matrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix} * A$$

With respect to every point in the picture, the subsequent slope approximations can be combined to determine the gradient magnitude, utilizing:

$$G = \sqrt{G_x^2 + G_y^2} \tag{4}$$

By considering the above information, gradient direction can be calculated as follows:

$$\theta = a\tan 2\left(\frac{G_y}{G_x}\right) \tag{5}$$

**Roberts cross filter:** In Robert's operator the input image is convolved with 2*2 kernels as stated below:

$$G_x = \begin{matrix} +1 & 0 \\ 0 & -1 \end{matrix} * A,$$

$$G_y = \begin{matrix} 0 & +1 \\ -1 & 0 \end{matrix} * A$$

Let I(x, y) be a point in the source picture and $G_x$ (x, y) be a point in a picture framed by convolving with the first kernel piece and $G_y$ (x, y) be a point in a picture shaped by convolving with the second kernel portion (Mageswari *et al*., 2013). The angle can then be characterized as:

$$\nabla = I(x,y) = G(x,y) = \sqrt{G_x^2 + G_y^2} \qquad (6)$$

The direction of the gradient can also be defined as follows:

$$\theta(x,y) = \arctan\left(\frac{G_y(x,y)}{G_x(x,y)}\right) - \frac{3\pi}{4} \qquad (7)$$

**Implementation:** The implementation is an important phase in software development. Implementation refers to conversion of system design to an operation. In software engineering, an implementation is a realization of a technical specification or algorithm as a program, programming segment or other PC framework through PC programming and arrangement.

The experiment compares the speed up achieved by processing various images using image processing algorithms on single core CPU, multiple-core CPU and GPU (Karimi *et al*., 2010).

**Method I; using single-core CPU:** This method implements the application in serial manner, where the instructions are executed one by one and it notes down the time required to implement this:

- Allocate memory on CPU: h_resultPixels, h_pixels, srcPath, h_ResultPath
- Load the PGM image
- Detect the edges of the images using sobel filter and save the image
- Repeat the steps for different sizes of image and compare the speed up achieved

**Method II; using multi-core CPU:** This uses multiple processing elements simultaneously to solve a problem. Multi-cores offer potential to compute more efficiently. OpenMP directives are used as programming framework. Here the problem is divided into separate pieces that can be solved simultaneously; also each piece of the problem is additionally separated to series of instructions. Instructions from each part execute at the same time on different CPU cores. With the imbedded compiler instruction #pragma_omp_parallel for, we can set the number of reading threads:

- Allocate memory on CPU: h_resultPixels, h_pixels, srcPath, h_ResultPath
- Load the PGM image
- Divide the work among multiple threads
- Detect the edges of the images using sobel filter and save the resultant image
- Repeat the steps for different sizes of image and compare the speed up achieved

**Method III; parallel execution using GPU:** This uses hundreds of GPU cores simultaneously to solve a problem. This method gives better performance than CPU, since CPU has less number of threads compared to GPU. Here CUDA is used as programming frame work. Compute Unified Device Architecture (CUDA) was developed by NVIDIA. The following steps need to be performed:

- Allocate memory on host: d_resultPixels, h_resultPixels, h_pixels, srcPath, h_ResultPath
- Allocate memory on device: d_pixels, d_resultPixels, d_ResultPath
- Load the PGM image
- Transfer h_pixels from host memory to the d_ pixels on device memory
- Detect the edges of the images using sobel filter on device and save the image
- Transfer results from the GPU back to the host
- Repeat the steps for different sizes of images and compare the speed up achieved

## RESULTS AND DISCUSSION

Our experiment tests the power of parallel computing against computation needed for processing of larger images. The project is composed with sequential C, OpenMP and CUDA interface utilizing Visual Studio C++ 2010, executed on GeForce 820M graphic card. The tests are completed individually on a quad-core processor PC. Various sizes of images were considered and the corresponding execution times were noted.

The input image is shown in Fig. 1a. Once the edge detection is performed using Sobel, Prewitt and Robert's Cross filter, the output produces is Fig. 1b-d, respectively. To decide productivity of parallel program, select gray scale pictures with different sizes as test

Fig. 1: a) Input image; b) Sobel edge detection; c) Robert edge detection and d) Prewitt edge detection

Table 1: Sobel filter

| Image size (width*height) | Execution time in CPU (msec) | Execution time in openMP (msec) | Execution time in GPU (msec) |
|---|---|---|---|
| 128*128 | 0.0 | 0.000 | 0.142800 |
| 256*256 | 32.0 | 15.625 | 0.465800 |
| 512*512 | 64.0 | 31.250 | 1.726992 |
| 1024*1024 | 128.0 | 62.500 | 7.158000 |
| 1600*1200 | 192.0 | 93.750 | 13.005800 |
| 4000*3000 | 1168.0 | 546.875 | 80.920000 |

Table 2: Prewitt filter

| Image size (width*height) | Execution time in CPU (msec) | Execution time in openMP (msec) | Execution time in GPU (m sec) |
|---|---|---|---|
| 128*128 | 0.0 | 1.9531 | 0.145664 |
| 256*256 | 0.0 | 7.812 | 0.523000 |
| 512*512 | 32.0 | 25.3906 | 1.942100 |
| 1024*1024 | 110.0 | 48.8281 | 7.508300 |
| 1600*1200 | 188.0 | 91.7968 | 13.792000 |
| 4000*3000 | 1204.0 | 693.359 | 85.475000 |

Table 3: Robert cross filter

| Image size (width*height) | Execution time in CPU (msec) | Execution time in openMP (msec) | Execution time in GPU (msec) |
|---|---|---|---|
| 128*128 | 0.0 | 0.0 | 0.077312 |
| 256*256 | 16.0 | 5.859 | 0.286144 |
| 512*512 | 46.0 | 13.6718 | 1.124500 |
| 1024*1024 | 94.0 | 48.8181 | 4.478400 |
| 1600*1200 | 172.0 | 70.3125 | 8.189600 |
| 4000*3000 | 968.0 | 429.687 | 50.99142 |

objects. Utilizing single-core CPU, multiple-core CPU and GPU (Karimi *et al*., 2010), execution comparison for sobel edge detection, prewitt edge detection and robert-cross edge detection applications is done as results are depicted in the tables below. Table 1 shows the execution time needed to apply sobel filter. Similarly Table 2 and 3 show the execution time required on various thread distributions for the execution of Prewitt and Robert's cross filter. Figure 2 graphically depicts the above results.

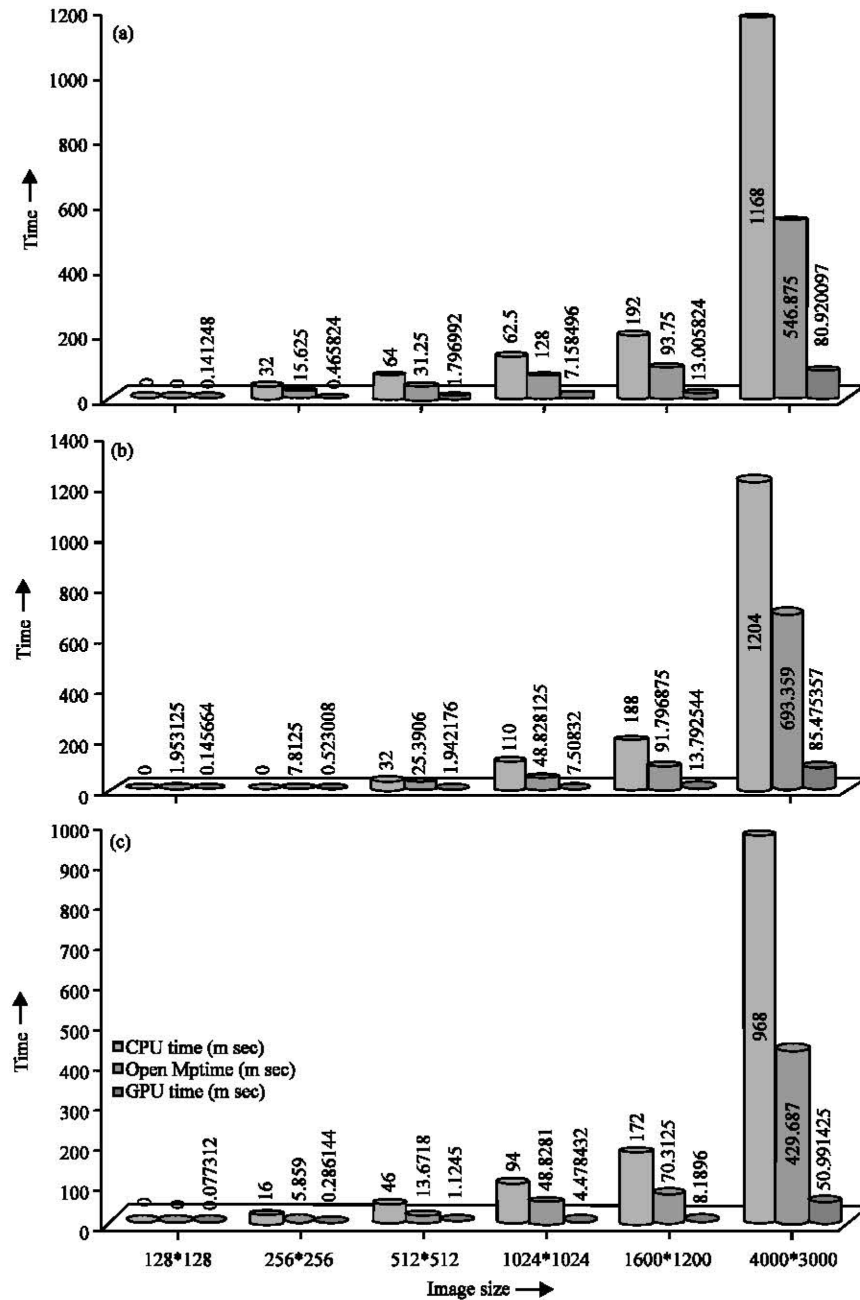


Fig. 2: Comparision of performance on CPU, Multi-core CPU and GPU using; a) Sobel edge detection filter; b) Prewitt edge detection and c) Robert cross edge detection filter

## CONCLUSION

In this study, a mechanism for implementation of edge detection filters is proposed. Edges are detected using various thread distributions and the speedup is noted, test contrasts the execution of GPU and single-core and multiple-core CPU for three applications. The outcomes from the tables and the diagrams demonstrates that multiple-core CPU has preferable execution over the single-core CPU. However a GPU framework has surpassed them with much better execution over CPU with respect to image processing applications. Three parallel image processing algorithms namely Sobel edge detection, Prewitt edge detection, Robert cross edge detection are presented and implemented on GPU and compared with the sequential implementation based on

single core CPU and multiple-core CPU. Execution outcomes demonstrate that noteworthy speedup is accomplished and the speedup increments with picture size expanding. Location of edges can get speedup compared with CPU-based executions. Clearly, GPU gives a novel and proficient quickening system for picture processing and is less expensive as compared to hardware usage. It may be noted that the edges detected using sobel filter are more precise and clear. This is due to the selection of the appropriate kernel. Considering the above fact and the test results, it can be said that sobel edge detection serves the best and can be used for intensive real time applications such as medical images and so on where detection of every edge is of at most importance.

When computation is done in GPU, CPU remains idle. In future this could be overcome by giving part of work to CPU while computation is carried out on GPU in parallel. Also the above said edge detection algorithms could be executed on OpenCL and MPI and the relative performance could be measured.

## REFERENCES


Arnautovic, M., M. Curic, E. Dolamic and N. Nosovic, 2013. Parallelization of the ant colony optimization for the shortest path problem using OpenMp and CUDA. Proceedings of the 2013 36th International Conference on Information and Communication Technology Electronics and Microelectronics (MIPRO), May 20-24, 2013, IEEE, Opatija, Croatia, ISBN:978-953-233-073-1, pp: 1273-1277.

Bhardwaj, S. and A. Mittal, 2012. A survey on various edge detector techniques. Procedia Technol., 4: 220-226.

Burkitt, M., D. Walker, D.M. Romano and A. Fazeli, 2010. Using the GPU and Multi-core CPU to generate a 3D oviduct through feature extraction from histology slides. Proceedings of the 2010 Ninth International Workshop on Parallel and Distributed Methods in Verification and Second International Workshop on High Performance Computational Systems Biology, September 30-October 1, 2010, IEEE, Enschede, Netherlands, ISBN:978-0-7695-4265-2, pp: 78-87.

Chouchene, M., F.E. Sayadi, Y. Said, M. Atri and R. Tourki, 2014. Efficient implementation of sobel edge detection algorithm on CPU, GPU and FPGA. Intl. J. Adv. Media Commun., 5: 105-117.

Dawson, L. and I.A. Stewart, 2014. Accelerating ant colony optimization-based edge detection on the GPU using CUDA. Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC), July 6-11, 2014, IEEE, Beijing, China, ISBN:978-1-4799-1488-3, pp: 1736-1743.

Gupta, S. and M.R. Babu, 2011. Performance analysis of GPU compared to single-core and multi-core CPU for natural language applications. Intl. J. Adv. Comput. Sci. Appl., 2: 50-53.

Karimi, K., N.G. Dickson and F. Hamze, 2010. A performance comparison of CUDA and OpenCL. Simone Foundation, Paris, France.

Khan, F.G., O.U. Khan, B. Montrucchio and P. Giaccone, 2011. Analysis of fast parallel sorting algorithms for GPU architectures. Proceedings of the Conference on Frontiers of Information Technology (FIT), December 19-21, 2011, IEEE, Islamabad, Pakistan, ISBN:978-1-4673-0209-8, pp: 173-178.

Lu, W., W. Zhigang and C. Zixing, 2012. Research on parallel algorithm of salient features extraction for nature scene recognition. Proceedings of the 2012 International Conference on Computer Distributed Control and Intelligent Environmental Monitoring (CDCIEM), March 5-6, 2012, IEEE, Hunan, China, ISBN:978-1-4673-0458-0, pp: 564-567.

Mageswari, S.U., M. Sridevi and C. Mala, 2013. An experimental study and analysis of different image segmentation techniques. Procedia Eng., 64: 36-45.

Wang, Y., Z. Feng, H. Guo, C. He and Y. Yang, 2009. Scene recognition acceleration using cuda and openmp. Proceedings of the 2009 1st International Conference on Information Science and Engineering (ICISE), December 26-28, 2009, IEEE, Nanjing, China, ISBN:978-1-4244-4909-5,