Humerus X-ray Analysis Project with MPI.NET

Rattanaphan Boonbutra, Tom Delaney, Department of Computer Science, Hood College.

Abstract— it is time consuming for pixel level image processing. Sequential programming can not take advantage of today's multi-core architecture. We propose a Massaging-Passing based parallel programming scheme to accelerate this process. MPI.NET was chosen because of its high-level binding capabilities of popular C# programming language on widely available Microsoft Windows platforms. We obtained close to linear speed-up with two nodes participating in the computation in mading broken bone diagnoses.

Index Terms—C#.NET, MPI.NET, X-ray film analysis, radio medical Analysis.

1 Introduction

THIS project is based on a Perception AI project which involves the use of algorithms to analyze x-rays and determine if the Humerus bone is broken. The program analyzes each pixel of the x-ray which is time-consuming to the point that it becomes problematic to medical diagnosticians awaiting the results. Parallel programming can accelerate this processing of x-ray images and is able to provide acceptable diagnostic rates to such complex calculations.

As the analysis program is written in C#.NET and the Object-Oriented Message Parsing Interface (OOMPI) only supports C, C++, and FORTRAN, a suitable substitute was required. The high-level C# bindings to MPI are based on the interfaces in the OOMPI C++ library [1]. The library that this project uses is MPI.NET. MPI.NET allows for the implementation of the MPI features in C#, and in this case allows us to optimize the code used to analyze the edges of the bone.

Image processing techniques are classified into two broad families. These are region-based and contour-based approaches [2]. Region-based approaches try to find sets of image pixels corresponding to coherent image properties such as brightness, color and texture [2]. Contour-based approaches usually start first with edge detection, followed by a linking process that seeks to exploit curvilinear continuity [2].

MPI.NET is a high-performance and simple-to-use implementation of (MPI) for Microsoft's .NET environment. MPI is the defacto standard for writing parallel programs running on distributed memory systems, such as a computer clusters, and is widely im-plemented. Most MPI implementations provide support for writing MPI programs in C, C++, and FORTRAN. MPI.NET provides support for all the .NET languages, especially C#, and includes significant extensions (such as automatic serialization of objects) that make it far easier to build parallel programs that run on clusters [3].

2 MPI.NET

MPI.NET (high-level) binding in C#.NET is based on the interface to the OOMPI C++ library [4]. MPI.NET tries to preserve the port and messaging features of OOMPI.
The port identifies the source or destination of the message. It includes a com-municator to use for the send or receive operations and also the rank within the communicator.

A message in MPI.NET is a typed buffer. It includes the starting address, length, and MPI data type for the data buffer to send or receive.

MPI.NET creates an interface and better adheres to the standard naming con-ventions used in the C# library than those of other libraries. A constant such as MPI_STATUS becomes MPI.status in the high-level C# bindings. In addition, it provides the class and properties for MPI.

3 Basic Information About The Computer Running Program

3.1 System

1. Processor: Intel® Core™, i3-2370M CPU @ 2.40GH
2. Memory (RAM): 6 GB
3. System type: 64-bit Operating system

3.2 Operating System

Operating System: Windows / Home Premium Service Pack 1

3.3 Execute program

Windows HPC pack 2008 R2

4 Implementing Parallel Computing (MPI.NET) For The Humerus X-ray Analysis Project

The Contour Analysis code sees an improvement in performance by using MPI.NET. The contour analysis process examines extremely large binary data which can be very time-consuming. It is necessary to threshold the grayscale x-ray image be-fore inputting it to the x-ray analysis program.

Fig. 1 on the left: before thresholding. On the right: Humerus image after thresholding.

After thresholding the image is smooth and free of most noise. Using this output, the x-ray analysis program will find the image's contours.

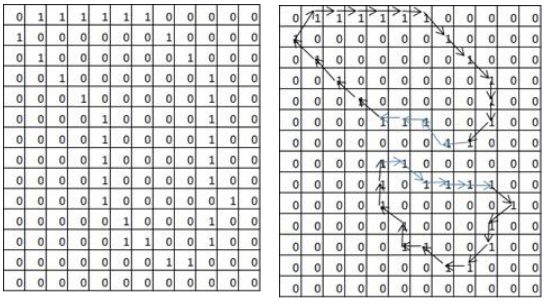

Fig. 2 shows contour vectors of broken bone.

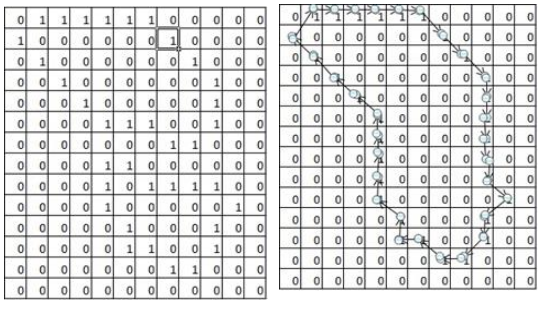

Fig. 3 shows contour vectors of non broken bone.

Sequential programming starts scanning from top left to the bottom right. It takes a long time to go through each pixel for a step-by-step analysis before result is forwarded to the contour image recognition process.

Parallel image processing using MPI reduces the scanning time. MPI is used to distribute data between processors and allows for inter-processor communication. With the message-passing model, we are able to work with very large sets of data without being restricted by the size of the global shared memory pool [8]. For example if we use 8 nodes for scanning the image, it will divide the work into 8 parts and split the image as illustrated in Fig 8, below. The data will be distributed to each individual node for working on the contour analysis process [9].

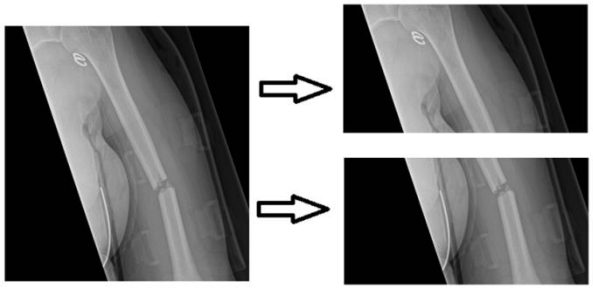

Fig. 4 shows the idea of using parallel image processing algorithm

Data parallelism distributes the data of the image. Many image processing operations have locality of reference (segmentation, filtering, distance transforms, etc.).

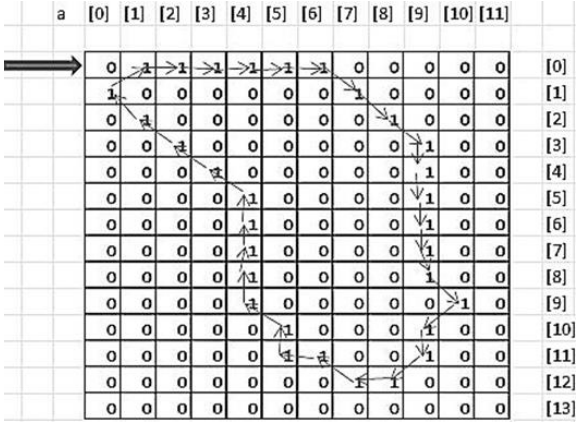

Fig. 5 non-parrallelized pixel processing

Fig 5 shows an image processing algorithm with contour analysis without paral-lel computing. The size of the image is 12x14 = 168 pixels. Each pixel is scaned 8 times, so the total scan time is 168x8 = 1,344 cycles. Assuming it takes 1 msec. per cy-cle, it will take a total of 1,344 x 1(ms) = 1.344 (seconds).

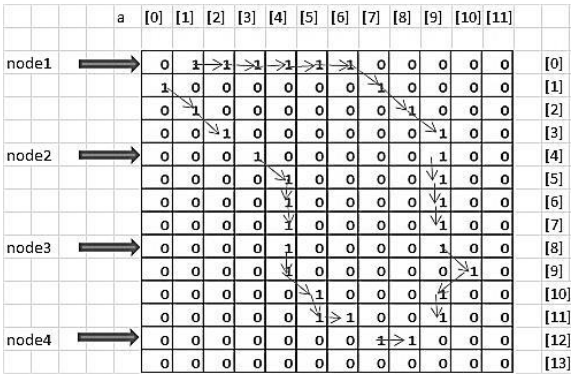

Fig. 6 parrallelized pixel processing

Fig 6, shows the image processing algorithm using contour analysis and parallel computing. Again, the size of the image is 12x14 = 168 pixels. Each node scans 12x4=48 pixels and each pixel is scaned 8 times, so the total scan takes 48x8 = 348 cy-cles. At 1 msec. per scan, it will take 348 x 1 (ms) = 0.348 seconds.

The benefit of parallel image processing, as demonstrated is that it provides fast-er processing of the templates containing the contour of the image.

The boundary scanning process adds each subset of the contour and connects them together. For example, the starting point of contour 1 is a[0][1], illustrated in Fig. 11. The set of boundary scanning areas will be {a[1][0], a[1][1], a[1][2]}. After searching for the starting point at every contour, we will find contour2 has a[1][0] as a starting point. Connect set of contours together and continue searching for a subset of a contour until meeting the end of contour1. Then check if sum of Vector Contour is 0. See fig. 7.

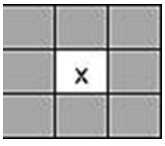

Fig. 7 shows boundary scanning area in gray color

As the program scans strating from row 0, it sets the value of the pixels already scanned to 0 so as to indicate that these points should not be scanned again. Next this current pixel's details are forwarded to the boundary scan process.

Boundary scanning scans left bottom, center bottom, and right bottom. The char-acteristic of the Humerus bone has a slope, so the boundary scanning process scans the pixel below the current x postion. The pixel counter is incremented and the program keeps looping until it can not find a connection to the next pixel.

While the program counts the pixels that it has processed, it compares the values to see if the pixel being analyized is equal to the image height in pixels and if so stops and counts the results as 1 parallel line.

The end of the program will check if two lines are parallel, and if they are results consider the image being scanned to include a broken bone.

5 Implementing MPI.NET with C# code in Humerus Analysis Project

The x-ray analysis program discussed, takes advantage of the parallel capabilities of MPI.NET, after the initial pre-processed stage has completed. The bit analysis portion is the more time intensive portion of the analysis, as this is where the more in-tensive contour analysis occurs. The parallel programming section produces subsets of the original contour as seen in Fig. 7. The Boundary scanning process connects the subsets of contour together. In each node it works independly on each part of the im-age. See from the code example below.

```
node1    contour1 ➤ {a[0][1], a[0][2],a[0][3],a[0][4],a[0][5],a[0][6],a[1][7],a[2][8],a[3][9],\0}
         contour2 ➤ {a[1][0],a[2][1],a[3][2],\0}

node2    contour3 ➤ {a[4][3],a[5][4],a[6][4],a[7][4],\0}
         contour4 ➤ {a[4][9],a[5][9],a[6][9],a[7][9],\0}

node3    contour5 ➤ {a[8][4],a[9][4],a[10][5],a[11][5],a[11][6],\0}
         contour6 ➤ {a[8][9],a[9][10],a[10][9],a[11],[9],\0}

node4    contour7 ➤ {a[12][7],a[12][8],\0}
```

Fig. 8 shows contour of each node

```
using (new MPI.Environment(ref args)) {
 Intracommunicator comm = Communicator.world;
  …
 Bitmap New_Bitmap = new Bitmap(width, heigh / comm.Size);
 …
  for (int i = (heigh / comm.Size) * comm.Rank; i < (heigh / comm.Size) *
(comm.Rank+1); i++) {
      for (int j = 0; j < width; j++)  {
       a = (j + Start_Pixel_x);
        b = (i + Start_Pixel_y);
       Color pixel = bitmap.GetPixel(a, b);
         if (pixel.R == 255 && pixel.G == 255 && pixel.B == 255)  {
         count_pixel++;
         bitmap.SetPixel(a, b, Vector_Color);
         New_Bitmap.SetPixel(j,m, Color.Red);
         Find_Boundary_List(a, b, New_Bitmap, comm,m); …}
      }}}
```

CODE 2. EXAMPLE C#.

6 Setup Dependencies And Running Of The Program With Window HPC Pack 2008 R2

6.1 Install depencies program and Install HPC (High Performance Computing) Pack 2008 R2 in order.

1. Set up .Net 4.5 (dotnetfx45_full_setup.exe)
2. Install Microsoft HPC Pack 2008 R2 (HPC Pack 2008 R2 with SP4)
3. Install MSMPI (mpi_x64/x86.msi, it will give HPC pack 2008 mpiexec.exe)
4. Install MPI.NET1.0.0
5. Install VS C++; vcredist_x86/x64
6. Install emgucv-windows-universal-gpu 2.4.9.1847(OpenCV_Emgu)
7. Run code executable as illustrated in Figure 9, below.

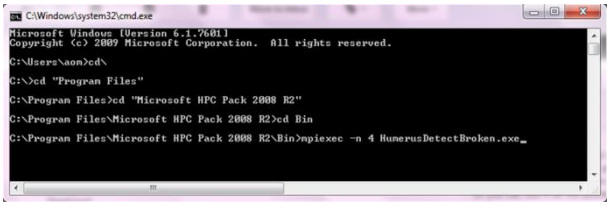

Fig. 9 running Humerus Analysis specifying 4 nodes.

Note: Error "Threw Emgu.cv.Invoke" use DependencyWalker to find what is evex-tern.dll want, then move that file's (.dll) to ./Microsoft HPC Pack 2008 R2/bin

6.2 Tracing MPI.

1. Install viewer "Microsoft Windows Performance Toolkit"
2. Share and set permission the folder which will college mpi_trace.etl file
3. Run command prompt command as administrator (Ctrl+Shift+Enter)
4. Execution the application.exe by using command “mpiexec –trace –cores 2 –n 2 HumerusDetectBroken.exe”
5. Check traces file in (folder shared)/mpi_trace.etl

Fig. 10 shows file .etl from trace MPI

6. Double click mpi_trace.etl it will show up as Figure 11.

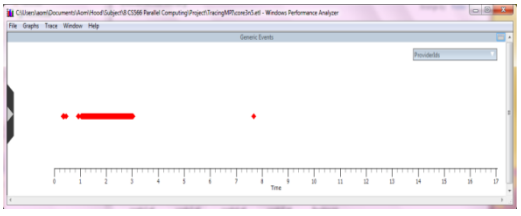

Fig. 11

7. Right-click on the graph, then click “Simple Summary Tables” follow as Figure 12.

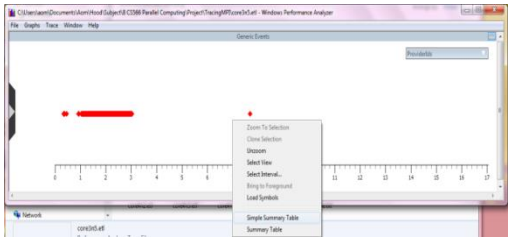

Fig. 12

8. The summary data table will show up as Figure 13

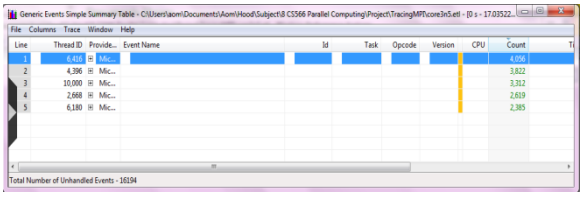

Fig. 13

9. The summary data table shows all of the data of MPI running as Figure 14

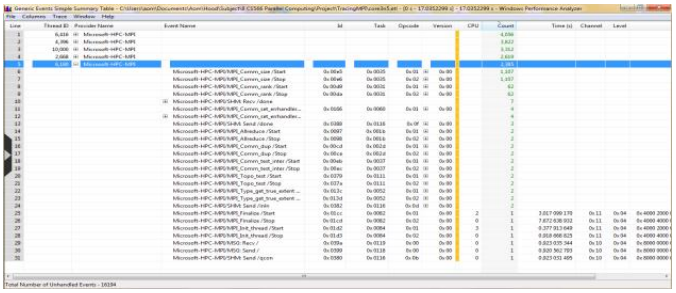

Fig. 14

7 Results

$$Sp = T1/Tp \quad (1)$$

Where:
Sp is the speedup
p is the number of processors
T1 is the execution time of the sequential algorithm (core1, n = 1)
Tp is the execution time of the parallel algorithm with p processors

From the result in Figure 15, the time consumed is decreased at size of node is 2 for every core. On the other hand speed will be increased at node is 2 for every core. The highs peak of speed up at core 2, node 2 and after it has a highest peak it goes down.

core	n	time	Speed up
1	1	1671	1
	2	1444	1.157202
2	1	1611	1.037244
	2	1370	1.219708
3	1	1550	1.078065
	2	1418	1.17842
4	1	1678	0.995828
	2	1409	1.185947

Fig. 15 Data table of time consume and speed up for each core and node changing.

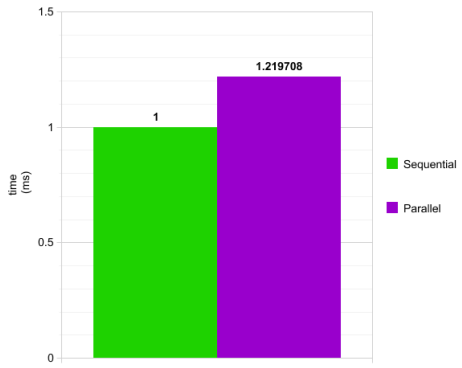

Fig. 16 Improvement graphs.

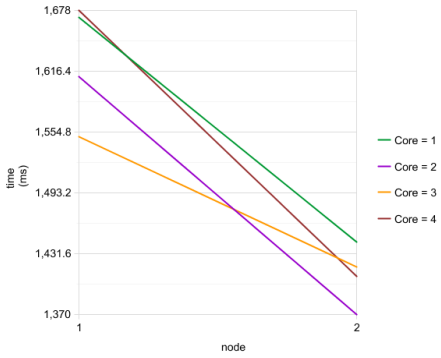

Fig. 17 Time consuming graph

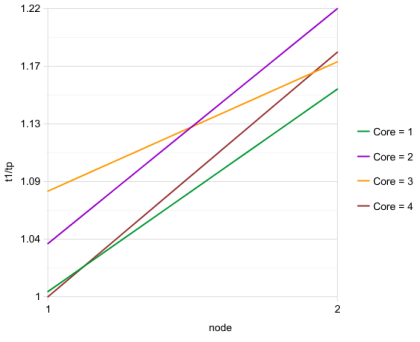

Fig. 18 Speed up graph

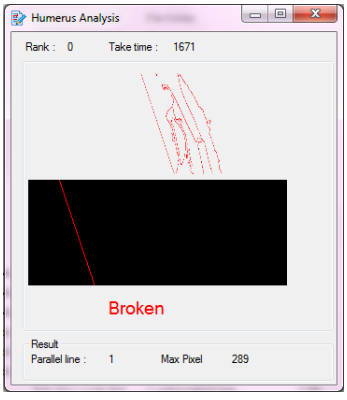

Fig. 19 Output of program when it rus as sequential computing

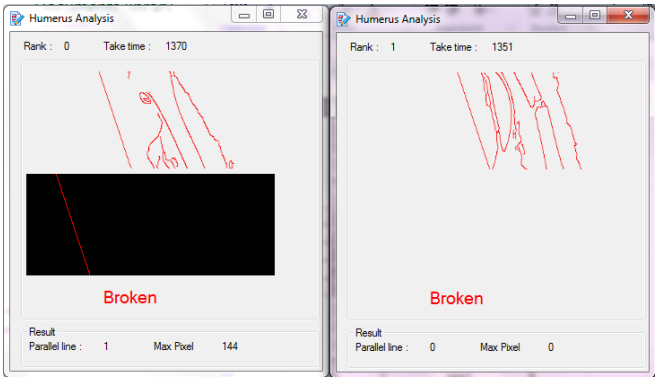

Fig. 20 Output of program when runs using parallel computing (cores=2, nodes=2)

7 Problem

The performance of running a parallel program in a single machine is that the parrallelization advantages solely rely on utilization of the multiple cores of this system. The program demonstrated was on a machine having an Intel® Core™, i3-2370M CPU @ 2.40GH and 6 GB (RAM). We did indeed see that using 2 cores did provide the best results. Unfortunately, this system is limited by the CPU cores, and no further speed gains could be realized.

Running parallel computing using Windows HPC cluster server 2008 R2 can solve this problem. A single machine could have gains in more cores, and further improve-ments could be realized, but the big advances would come from a fully configured MPI.net topology. To set up such a topology, the head machine needs to have at lease 3 networks cards, interconnected to a central network, and be configured to communicate with the rest of MPI networked machines.

The workstation machines are the machines that connect with MPI headnode. There are assigned jobs from the headnode and the results in turn are sent back to the head node. The workstation machines also need to be running 64-bit Windows as the HPC 2008 client is required.

8 Future Work

Implement HPC cluster server 2008 R2 cluster to further accelerate the performance gains and take full advantage of parallel computing. HPC permits running the program through the headnode, and can assign the job to run across multiple nodes among all available machines.

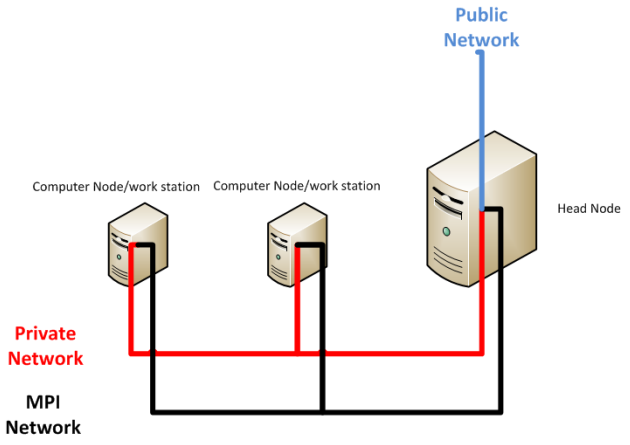

Fig. 21 HPC Cluster Topology for MPI

To set up Windows HPC cluster, we need at least 2 computers. One will be a head node to provide jobs to computer nodes. Windows HPC Server 2008 R2 requires 64-bit for both the head node and the computer nodes. The server needs to have at least 3 network cards for internet, intranet (private network), and MPI network.

Using VMware on the head node provides a plan where we could install windows HPC Server 2008 R2 and create 3 virtual network cards. The computer nodes do not need to be servers and could be either server or workstations.

Setting up the HPC server requires setting up the server roles and also the domain for the head node as an active directory domain controller. After it is setup, the HPC cluster will be configured for the MPI network topology as seen is Fig 21. Configuring the Naming of new nodes is the next step from the HPC server. There are 3 ways to setup the nodes. Nodes can be deployed from scratch, importing or a node specified by an XML file, or added as a preconfigured node set from the head node (server).

After the head node and the client nodes are set, the head node provides a job to the client node(s) by using HPC job manager. Similarly the client node uses the HPC job manager to connect back to the head node and respond with results [11][12].

9 Conclusion

In a 2-core 2-node configuration, the program runs the fastest and consumes less time. Increasing either the number of cores or nodes, the program runs slower. Having more cores or nodes does not directly translate into a more efficient execution of the program. In fact, execution efficiency also depends on things like the size of the pro-gram, the complexity of the program and the configuration of the executing computer.

Acknowledgment

Thanks to Dr. Xinlian Liu, Professor of Computer Science at Hood College, MD. Thanks Mr. Niwat Waropas Senior staff Engineering, Advance Automation Engineering at Seagate Technology Thailand.

References

[1] J. Wilcock, A. Lumsdaine, A. Robison, Using MPI with C# and the Common Language Infra-structure., [Online]. Available: ftp://www.cs.indiana.edu/pub/techreports/TR570.pdf

[2] J.Malik, S.Belongie, T.Leung and J.Shi, Contour and Texture Anlysis for Image Segmenta-tion. [Online]. Available: http://www.eng.utah.edu/~bresee/compvision/files /MalikBLS.pdf

[3] The Trustees of Indiana University, (2013), MPI.NET High-Perfomance c# Library for Message Passing. [Online]. Available: http://www.osl.iu.edu/research/mpi.net/

[4] P.Torgashov, Contour Analysis for Image Recognition in C#. [Online]. Available: http://www.codeproject.com/Articles/196168 /Contour-Analysis-for-Image-Recognition-in-C

[5] S.Chatzichristofis, C# Tutorials. [Online]. Available: http://savvash.blogspot.com/p /c-tutorials.html

[6] D.Gregor, A.Lumsdaine, MPI.NET High-Performance Message Passing in C# and .NET. [Online]. Available: http://research.ihost.com/ppopp08/presentations/gregor-mpi-net.pdf

[7] M.Podwysocki, Concurrency with MPI.NET. [Online]. Available: http://weblogs.asp.net/podwysocki/archive/2008/05/15/concurrency-with-mpi-in-net.aspx

[8] A.Jackson, A Parallel Algorithm For Fast Edge Detection On The Graphics Processing Unit. [Online]. Available : http://wwwx.cs.unc.edu/~jacksona/doc/thesis.pdf

[9] J.Squyres, A.Lumsdaine, R.Stevenson, A cluster-based parallel image processing toolkit. [Online]. Available: http://www.osl.iu.edu/publications/prints/1998/squyres98:_pipt.pdf

[10] W.Caarls, Parallel Image Processing. [Online]. Available: http://wouter.caarls.org /publications.html

[11] Microsoft Corporation, Windows HPC Server 2008 R2 Technical Library, (2010, October). [Online], Available: http://technet.microsoft.com/en-us/library /ee783547%28WS.10%29.aspx

[12] Microsoft Corporation, Remote Desktop Services in Windows Server 2008 R2, (2010, July 19). [Online], Avalilable: http://technet.microsoft.com/en-us/library /dd647502%28WS.10%29.aspx