Humerus X-ray Analysis Project with MPI.NET

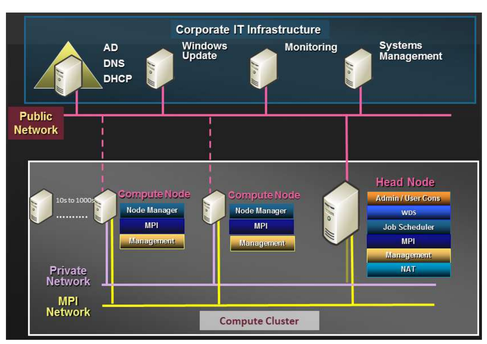
Rattanaphan Boonbutra, Tom Delaney, Department of Computer Science, Hood College

- Analysis of X-rays by perceptual AI (Artificial Intelligence) combined with multiprocessing algorithms illustrates speed improvements.
- Open MPI provides a message parsing interface which communicates complex processing and handling of subparts being analyzed
  - Speed improvements demonstrate rate improvements and make x-ray analysis practical for multi-processor algorithms utilization.

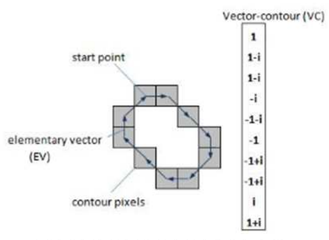
Abstract

In image processing, the time consumed to analyze every pixel of an image is a big problem. Even with multi-core computing, sequential programming is not effective for image processing. Parallel programming using the "Message Passing Interface" (MPI) accelerates this image processing and takes advantage of today's technologies. MPI.NET uses the high-level binding capabilities of the C# programming language and provides a MPI interface for distributed memory and parallel computing. The performance benefits provided by the C# bindings improves image processing with .NET, as the time spent analyzing each pixel decreases and we can increase the overall efficiency of the processing.

After thresholding the mage is smooth and free of most noise. The pre-processed x-ray is then further analyzed to define the image contours and search for discontinuous edges indicative of broken bones.



Microsoft's High Performance Computing (HPC) server can manage multiple interconnected clients and take advantage of networked computing power (uses MPI.net for inter-communications). In an environment where multiple systems are idle for much of the time, a distributed software package can uti-lize the full processing power of the network resources.



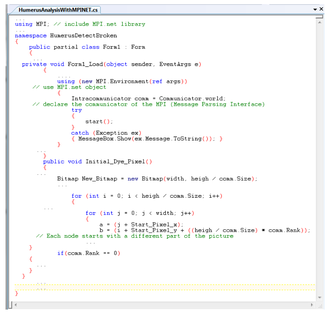
1+(1-i)+(1-i)+(-i)+(-1-i)+(-1)+(-1+i)+(-1+i)+(i)+(1+i) = 0 [1]

ftp://www.cs.indiana.edu/pub/techreports/TR570.pdf

Pixel analysis processed by Elementary Vector algorithms and stored under the Corresponding Set of Vector-contours.

Contour Analysis

1. The sum of Elementary Vectors (EV) of any closed contour is zero
2. Neither a parallel transposition (shifting) nor an orientation change (2-D rotation) of the source image, changes a contour-vector, because the contour is encoded, relative to the previous point as a sequence, until it again returns to the starting point.
3. Changing scale can be considered as multiplication of each EV and of the contour, to the scale factor.



```
using MPI; // include MPI.net library
namespace HumerusDetectBroken
{
    public partial class Form1 : Form
    {
private void Form1_Load(object sender, EventArgs e)
        {
            using (new MPI.Environment(ref args))
    // use MPI.net object
            {
                Intracommunicator comm = Comunicator.world;
    // declare the comunicator of the MPI (Message Passing Interface)
                try
                {
                    start();
                }
                catch (Exception ex)
                { MessageBox.Show(ex.Message.ToString()); }
            }
        }
        public void Initial_Dye_Pixel()
        {
            Bitmap New_Bitmap = new Bitmap(width, heigh / comm.Size);
            ...
            for (int i = 0; i < heigh / comm.Size; i++)
            {
                ...
                for (int j = 0; j < width; j++)
                {
                    a = (j + Start_Pixel_x);
                    b = (i + Start_Pixel_y + ((heigh / comm.Size) * comm.Rank));
    // Each node starts with a different part of the picture
    }
            if(comm.Rank == 0)
    {
    ...
    }
    }
    ...
    ...
}
```

Code fragment incorporates MPI.net communications and multiple iteration of multi-node analysis.

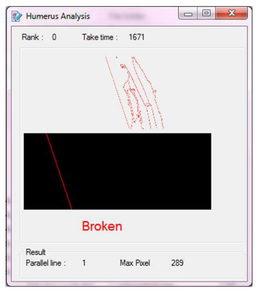
MPI.Net

MPI.NET (high-level) binding in C#.NET is based on an interface to the "Object Oriented Message Parsing Interface" (OOMPI) C++ library. 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 communicator 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 conventions 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 ad-dition, it provides the class and properties for MPI.



Single core analysis yields higher processing times

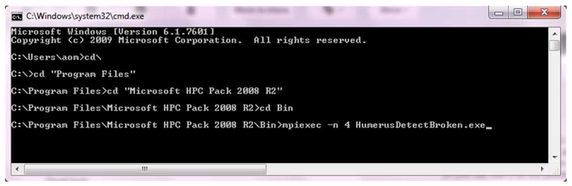
Running Program

From a centralized system, the ability to request running a job is readily available. From a command prompt window accessed from the Head Node, the full power of the network can be accessed.

Simply requesting a program's executable running with the command "mpiexec" provides access to specifying multiple node inclusions.

Here the term 'n-4' requests that 4 nodes operate together to work on processing the solution. The compute cluster has been preconfigured to provide distribution of the job among the resources available.

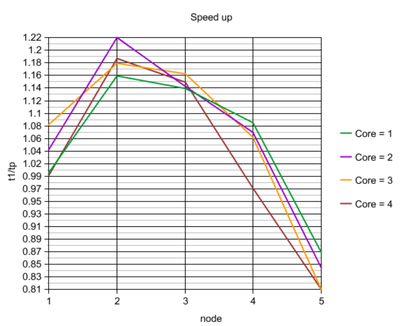
The MPI.net architecture was incorporated In the executable's design in order that the advantages of the distributed network could be fully realized.



Specifying multi-core nodes (-n 4) at startup of analysis

Standard processor architecture (non-MPI utilization)

A dual core computer is represented below, with various core inclusion specified. Since this specific system is only a dual core system, performance fall off when more cores are requested. Note however, the significant acceleration evident when the available 2 cores were requested, versus a single core run.

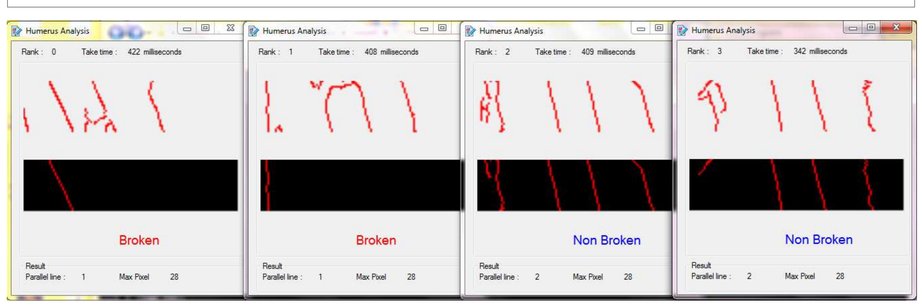


Simple 2-core analysis illustrates speed benefits over single-core comparison

Sequential processing vs. Parallel processing

Sequential programming, seen above, 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.

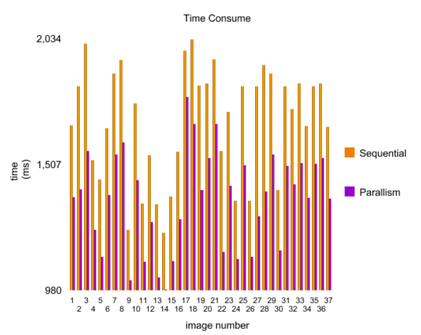
Parallel image processing using MPI reduces the scanning time. MPI is used to distribute data between processors and allows for inter-processor communication. With the message-passing model, we are able to work with very large sets of data without being restricted by the size of the global shared memory pool. For example if we use 4 nodes for scanning the image, it will divide the work into 4 parts and split the image as illustrated below. The data will be distributed to each individual node for working on the contour analysis process



Multi-core analysis breaks down analysis, with parallel individual time advantages.

Advantages of segmenting images for processing

If image analysis is performed from the top down, or with the use of a "sequential" processing approach, the load on the system is significant. By dividing the image in to multiple segments, a parallelized system is able to divide up the processing among the available cores.



Single core analysis yields higher processing times