

Colorado School of Mines, Field Session 2007

Los Alamos 2 – Parallel Search

Olen Davis, Bailey Kelly, Kari Macklin

Submitted June 22nd, 2007

Executive Summary

Los Alamos National Laboratory is a governmental nuclear weapons development facility. Additionally, the lab is a research facility for other scientific areas. They use large super computers to store their data and research. As data sets have grown increasingly large, the cost of searching the data has become progressively more expensive.

This project accomplishes three main ideas:

- 1) Develops a specific indexed search technique used for probing through a specific set of metadata. This is accomplished through the use of Google Desktop and its software development kit.
- 2) Creates its own dataset through the generation of custom tagged JPEGs. This done by creating the images using a Java program and then giving them unique tags through a C# program.
- 3) Creates a parallel computing environment to run searches on several computers at once. This is fulfilled by the use of a message passing interface, specifically, MPICH2.

The original goal of this project was to simulate large scale data searches by scanning multiple computers for JPEG image files. To accomplish this task, the project would combine parallelism using MPI with efficient search techniques using Google Desktop and compare these results with those found using Windows' old file search. The efficiency of these data searches would be numerically analyzed and represented graphically.

Unfortunately, the development took longer than expected, and that is why the project goals shifted. The searching and analyzing part of the project has been allocated as additional work after the completion of the Summer 2007 Field Session.

Introduction

Motivation and Client

Los Alamos National Laboratory is a governmental nuclear weapons development facility. Additionally, the lab is a research facility for other scientific areas. They use large super computers to store their data and research. As data sets have grown increasingly large, the cost of searching the data has become progressively more expensive.

They are looking for better ways to quickly and effectively search their data. Furthermore, this is why they have come to the Colorado School of Mines: to aide them in finding a solution to this very practical and important problem.

Our Project

Our project simulates searching metadata in parallel on a small cluster of computers. The goal of this project is to time-wise analyze search methods for efficiency. In turn, this will help LANL's efforts to minimize the cost of searching through their extensive amount of data.

In picking what method of searching we would use, LANL turned to software that was readily available and that came with a long list of credentials to boot: Google Desktop. This software is free and easily downloadable to anybody who has a computer and internet access. Many people are already aware of Google's web browsing dominance and high performance. Additionally, Google Desktop takes this same web browsing idea and applies it to the hard drive on your computer, quickly searching through all your files so you can find what you need much faster than clicking through your computer's browser. Google has raised the bar in searching techniques by utilizing a unique indexing capability that organizes your files ahead of time so they are quicker to search through. This is precisely why we are using Google Desktop.

We now have how we are going to search through large amounts of data, but now we need something to search through. In effort to make the searches more controlled, the project asks that we create our own dataset. JPEGs are a good example of metadata because they hold large amounts of searchable information within their structure.

The project also required some kind of message passing interface to allow the computers to talk to one another and perform the searches simultaneously. Once again, this came down to something that was readily available to the public, and this was the MPI library, MPICH2.

With the three main project ideas in place, it created a natural organization for the project and thus for this paper:

- 1) The JPEG Dataset
- 2) The Google Desktop Development

3) The MPICH2 Implementation of the Searches.

The Problem Statement

--To develop index search techniques for a generated JPEG dataset within a parallel computing environment--

Dataset

Introduction

The Parallel Search project requires a complex dataset containing metadata to be searched. In order to ensure the dataset would have metadata, the group and client decided to generate a dataset of JPEGs. The Specifications for JPEG files are produced by the Joint Photographic Experts Group and JPEGs are a common image format type [1]. A JPEG is a complex data type that contains far more information than the basic pixel data of the image. Some of the information stored in JPEGs includes important dates, color scheme information, and compression data. JPEGs are also capable of storing other types of data, including EXIF tags. Exchangeable Image File Format or EXIF tags can contain any kind of data but some of the most common tags are author, camera maker and comments [2]. Once the dataset is created it also needs to be able to be parsed out to obtain the information that has been added to it to be searched.

Requirements and Specifications

The requirements for the dataset were that it needed to have searchable metadata and since JPEGs are among the most common file types with complex metadata, they are a good choice to meet this requirement. Another requirement was that the dataset be large enough to provide an ample search. Due to time constraints and limited disk space, the dataset is not the 100 GB it was once thought it could be; instead it is 77 GB total. An external 250 GB USB hard drive was found to store the entire dataset and the dataset was also spread evenly across the eight computers.

In order to accomplish these requirements the Eclipse Java IDE was used as well as Visual Studio 2005 C# Express Edition. Also used was approximately 9.5 GB of the remaining 12-14 GB of the 8 computers' C drives as well as the external USB hard drive. Data about the distribution of the dataset can be found in Appendix A: Dataset Size per Computer.

Design

The dataset was generated using a combination of Java and C# programs. The Java program created images of random sizes using BufferedImage objects of random sizes that contained random Graphics2D shapes such as rectangles and ellipses with random colors, some with gradient color. This BufferedImage was then used to create a RenderedImage which could be written to a JPEG file [3]. In order to easily create files with different names, an incremented

value was used to assign a number to the title of the JPEG files. Since generating the basic images with Java, the Picture Generator program has been updated to include a graphical user interface which can be seen in Figure 1 below. The Picture Generator Setup Frame requests the number of images to be generated, the number assigned to the first JPEG generated, a directory location of for the images, the basic name for the images, height, width, and the extension for the picture files (i.e. – jpg). The JPEGs generated in this program are smaller than the average JPEG at only about 150-250 KB. This is because they were created from simple Graphics2D shapes rather than more complex image data like that from a camera where most JPEGs come from. The complete code for the Picture Generator can be found in the Appendix B: Picture Generator.

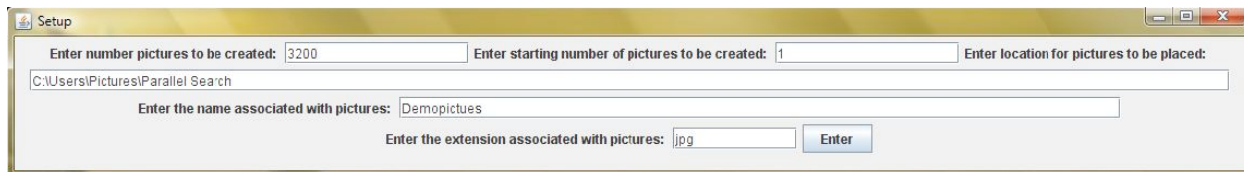


Figure 1: Picture Generator Setup Frame

After the images were generated, they were sent through various versions of the C# Tag Adder program which would add tags to the JPEGs and save the modified versions in a new file [4]. Different versions were created so that the JPEGs would have a variety of tags with different numbers of tags in different JPEG files. The code used for Tag Adder is contained in Appendix C: Tag Adder Versions.

An advanced version the Tag Adder program called Parallel Search Tag Adder was created to include a graphical user interface that would request the directory of the images and the extension of the original images as well as the extension the final pictures should have (i.e. - .jpg). Figure 2 shows how this program appears when run. The program adds all of the tags to each image then saves the Bitmap image to a new image file and deletes the original file. This program was developed at the end of dataset creation when it was decided that the dataset had many basic Java generated images that should take a smaller portion of the total dataset. The program was used to replace some of these basic JPEGs with more complex JPEGs that have 10 tags. Appendix D: Parallel Search Tag Adder contains the primary code for this program.

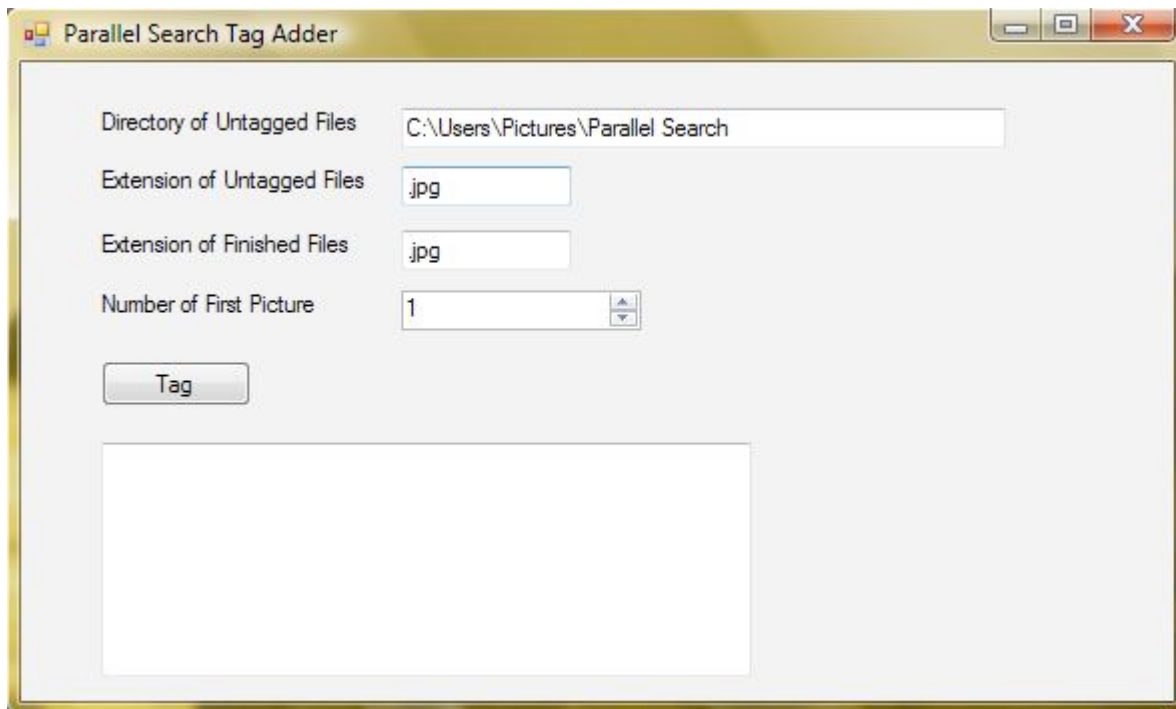


Figure 2: Tag Adder Setup Frame

As Figure 3 shows, ParallelSearchJPEGs is an application created in C# that parses JPEGs by using the C# image library that was used to parse tags in the JPEGs [4]. The program only handles the ASCII encoded tags, so the values of tags of other types such as the integer encoded tags are not handled properly by this program. Though, the program will provide the ID of the tags, the length, and the encoding type of the tag. This program was used to check that information was being properly added to the JPEGs and that the values of the tags could be obtained after being added to the JPEGs. The parsing code for this program is located in Appendix E: Parsing Code.

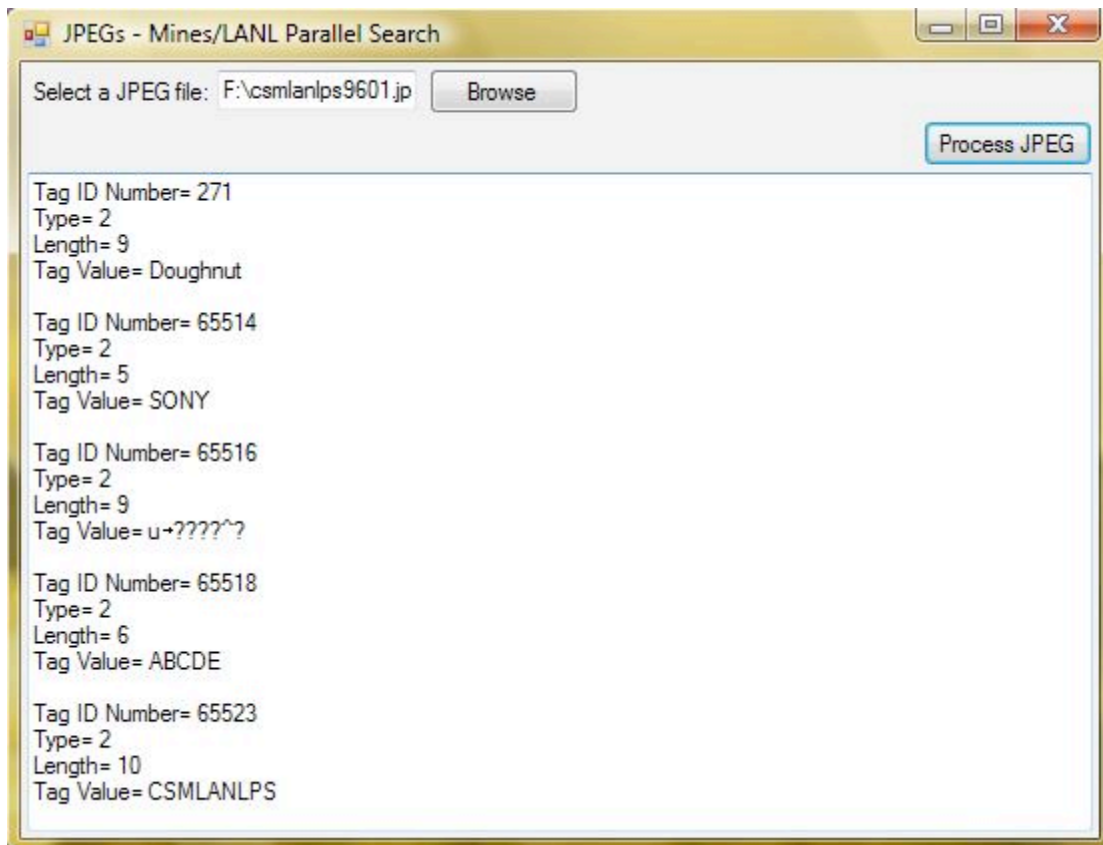


Figure 3: JPEG Parsed by ParraSearchJPEGs Program

Implementation

Dataset Generation Process

- 1) Images were generated using the Picture Generator program. In total, nearly 80,000 of these were created.
- 2) The JPEGs generated by using the Picture Generator program were sent through various versions of the Tag Adder program to create new JPEGs that contain modified EXIF data. Some pictures that had already been output from a Tag Adder program were then used on as the original versions of new pictures. Though, the majority of the dataset was created directly from the original group of almost 80,000 JPEGs.
- 3) The Parallel Search Tag Adder program was used to replace half of the original almost 80,000 JPEGs generated by the Picture Generator with new JPEGs that had their EXIF data modified to contain eight more tags.

Tag Description

Each tag has certain characteristics. There is an identification number, for example 271 is the identification number for the camera maker tag. When a JPEG parsing program searches for EXIF data in a JPEG it will search for this identification number. The tags also contain information about the length (usually in bytes) of the tag and the value of the tag [5]. The value for a camera maker tag might be Canon. For this project about a dozen different tags were added to the images, the only tags that were created that used an ID commonly associated with JPEGs

was the camera maker, Chrominance table and the Luminance table tags. The specific EXIF tag details are contained in the Dataset Programmer's Guide.

Chrominance table and Luminance tags were contained in all of the JPEGs since they were added by the Picture Generator program when they were created. Several tags had constant values throughout sets of pictures, but as requested by the client several other distribution types were used in the tags. There were two types of uniform distributions inside the tags, one which was supposed to choose with equal likelihood from one of the four values and another that would set the value based on random bytes that were not weighted in any way. Two types of weighted distributions selected from four values with differing probabilities of a value being selected. A highly discrete normal distribution was created where six different possible values were supposed to be chosen to create a semi-bell curve based on the percentages of 2.3%, 13.6%, and 34.1% [6]. A final distribution type was created out of integers produced by the Box-Muller Algorithm which creates normal distributions [7]. However, the algorithm generates pairs of values and only one of these pairs was added to the tag, making the distribution less Gaussian. Also, a logic error in the percentage based discrete distributions (weighted, normal, and uniform) skewed the percentages of the values in the dataset. This error has been resolved in the most recent versions of the Tag Adder programs.

Dataset Organization

The purpose of the dataset is to be a controlled test which verifies the Google Indexer's ability to properly index the given data. In order to do this, the dataset should have variation within it as well as some way to verify that the results are correct. For this project, a dataset of JPEGs was generated that consists of nine primary groups of data.

The first part of the dataset generated was the Blankjavaimages grouping. This is the most basic set of JPEGs and besides bulking up the dataset, these images ensure that the Google Desktop plug-in correctly handles images that lack the tags that it is indexing. The Blankjavaimages contain basic JPEG formatting, such as color type, the frame, time stamps, as well as two EXIF tags the Chrominance table and Luminance table. These JPEGs were generated using only the Picture Generator program and never had tags added to them from the Tag Adder program. If any search of the ASCII encoded tags returns one from this group, then we know an error has occurred in the search.

The second section of the dataset was initially called Taggedjavaimages, but these later became the CSMLANLPS images. The Taggedjavaimages were created the same way as Blankjavaimages and were basic JPEGs with the two basic tags. Along with Blankjavaimages, Taggedjavaimages were sent through as the various C# Add Tag programs to create the other groups by adding EXIF tags to the images and saving the images in new files with different names.

CSMLANLPS was the last group created and replaced the Taggedjavaimages group. The CSMLANLPS was created using the Parallel Search Tag Adder by adding EXIF tags to the Taggedjavaimages. The Taggedjavaimages group as then deleted by the program after it made the CSMLANLPS JPEGs. CSMLANLPS is the most complex set of JPEGs with ten EXIF tags in total. Each tag is encoded as ASCII characters, and the tags have various distributions such as weighted, uniform and normal distributions.

The Taggedjavaimage group is the third group of the final dataset. It was the first group of images to be tagged and contains the most basic tag with an ID of 66535. The tag in this group would have a single value throughout groups of JPEGs that were created together. The Taggedjavaimages group was used as the original files that were modified to produce the Taggedjavaimage files.

Also created early in the dataset generation process, group Twicetagged is a unique group that had two of three tags added to them. The 66535 tags were added to all of the JPEGs, and then either 65534 or 65533 were added to the image. The values of these tags wasn't varied within a set of images, instead the Tag Adder add one of these two tags based on an even probability.

This was the only group where the tags that were added were randomized, not the value of the tags. The program chose with an even chance between two different tags to add to the image.

The Camerataggedimage group verifies that the images are being tagged correctly so that other programs can open the JPEGs and read them correctly. Windows is capable of viewing several EXIF tags inside of JPEGs, one of which is the camera maker that the Camerataggedimage group contains.

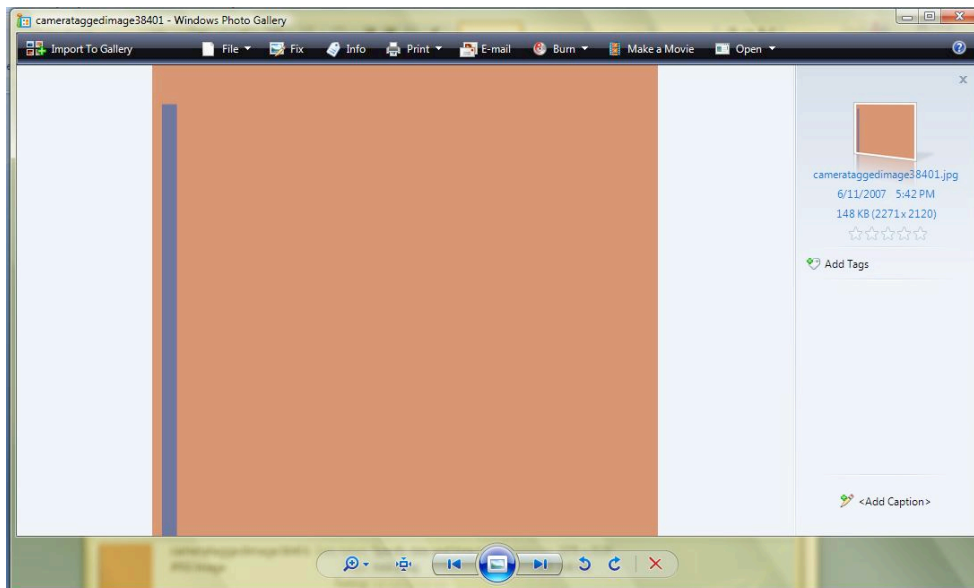


Figure 4: Example Camerataggedimage

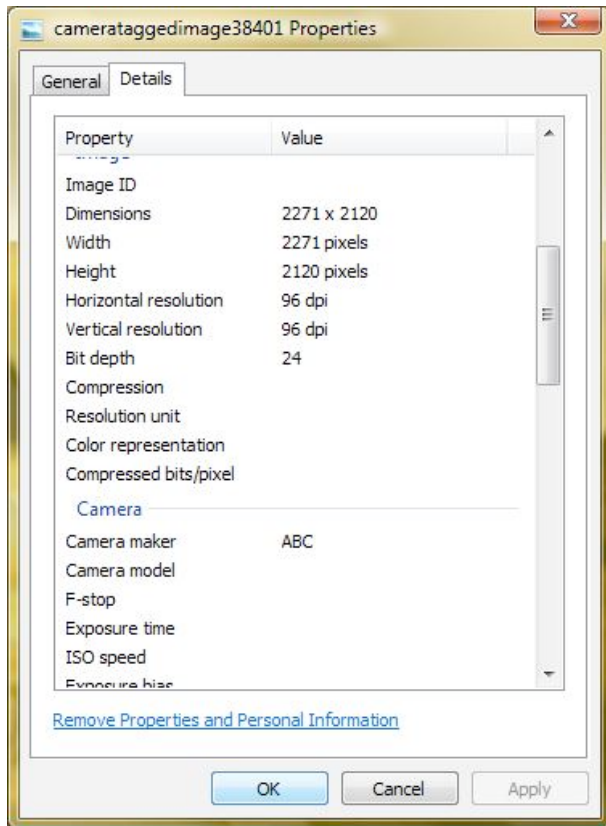


Figure 5: Windows Display of JPEG information

The above Figure 4 shows an example JPEG from the Camerataggedimage group. Figure 5 shows part of the information about the JPEG, specifically the camera maker tag which was encoded by one of the C# Tag Adder programs. This window can be displayed by right clicking on a JPEG, selecting properties and then the summary or detail tab. The camera maker can also be labeled as Equipment make.

The Uniformcamera group is similar to the Camerataggedimage group except that the images were assigned values in a uniform distribution. The values are chosen from one of four possibilities with a 25% chance of any value being assigned to the JPEG. Also contained in these tags were two tags which had the same value across the set.

Normal0xffff9 JPEGs were assigned values in a very basic normal distribution which chooses from six possible values that form a discrete version of the bell curve for the 0xffff9 tag. Tags 0xffff1 and 0xff55 were constant valued tags that were added to the JPEG set.

Uniformrandom0xffff8 group assigns the tag values based on random byte values. This is a uniform distribution, and any searches of this group would likely return only one result due to the number of possible combinations of byte values. Two constant valued tags were also added to this group.

The Normaltest0xffff7 group was an attempt to create a more realistic Normal distribution using the Box-Muller transformation of uniform random numbers into a normal distribution. The values ranged from approximately -200 to 200 with a center at 0. Other tags in the group is the camera make tag which is assigned random byte values, another constant valued tag, and a weighted tag where values a 10%, 15%, 25% and 50% chance of being assigned.

For specific information regarding the particular tags added to the JPEGs and distributions of these tags please see Appendix F: Tags by Group.

Scope and Project Progression

The dataset part of the project meets the minimum requirements for the project. It is a large dataset that contains metadata. Since it was created during the development stage there are several remnants of earlier JPEGs that lack the complexity of later JPEGs. The dataset should include some JPEGs that are at least similar to these earlier JPEGs as proof that the Google Desktop is capable of handling any JPEG. However, the dataset is lacking in a unifying trait that marks the JPEGs as part of the dataset. Such a trait was not part of the scope of the project but could be considered if the project were continued.

Final Dataset Specifications

The final dataset is 77 GB in size and has 364,000 JPEGs. The JPEGs have a basic JPEG data and from two to ten EXIF tags. For a complete listing the dataset specifications please see Appendix G: Dataset. The dataset has been spread across the eight lab computers supplied for the project and another copy is located on the external USB drive.

Conclusion and Future Directions

Creating this dataset has shown that generating the image part of the JPEG is the most time consuming process, while adding the tags to an existing JPEG takes less time. The dataset was being created while methods for generating JPEGs and adding tags were being tested causing most of the groups to be somewhat experimental. Some were successful groups such as the Camerataggedimage group, while others had characteristics that weren't used again while creating this dataset like the Twicetaggedimage group.

Future goals would be to create a single program which would be able to generate images and add various tags to that image which could then be saved as a JPEG. Possible input for the program could be such information as tag id, encoding type, possible tag values and type of distributions. This may be possible by investigating C# image classes to generate the image part of the JPEG or perhaps the lockbit functionality of C# which deals with rectangular sections of C# Bitmaps [8]. A few test JPGs were created in C#, but the process was slow because a pixel by pixel generation process was used [9]. These images also proved difficult to add a tag to which is an error that remains unresolved. C# appears to have a great potential in producing JPEGs of larger size as it was used to generate a JPEG that was 211 MB in size. Unfortunately, producing this JPEG took almost 40 minutes. The pixel by pixel C# JPEG generation code is located in the Appendix H: C# JPEG Generator.

A Java program utilizing the IIOMetadata class may also be able to add the tags to the Java generated images [10]. Java has some online examples for parsing JPEGs and tends to produce trees out of the metadata with each node being a tag. The parsing and accessing of tags within the Java language is a reasonable process, however little development has been done to set or append new tags into the JPEG.

Whichever language chosen, the final program should have some sort of randomness to it that will add different numbers of tags. This could be implemented similarly to the way in which the

Graphics2D objects were added to the BufferedImage, or perhaps some forms of weighting could be used which would add some tags more than others, preferably using tags that did not ID values of 0xFFxx since these tend to be reserved for other important JPEG tags [5]. Another future goal would be to generate JPEGs with a larger variety of sizes, perhaps with different compressions. Also, the organization of the images could be investigated by altering the directory structure in which the JPEGs are organized. In the future, the tags could possibly be encoded with other types besides ASCII byte arrays and the ASCII byte arrays could be formatted correctly to be terminated with a null value so that all programs correctly read the values in the tags. Two final ideas would be that instead of using the .jpg extension, the files would be saved with a special extension created for the project, for example .ps and that a special tag could be added to all of the pictures with a consistent value throughout the dataset to mark the JPEGs as part of the dataset.

Google Desktop Plugin Development

Introduction

Ultimately, this project has been divided into three implementation portions. The first is generating JPEGs, whether they be created or found. The second is implementing means for indexing and querying these JPEGs with Google Desktop. The third is executing and timing given queries in parallel across the eight computers. This portion of the report describes the motivation for and accomplishments made implementing that second portion of the project: the Google Desktop plugins. To that end, let us begin by describing the Google Desktop system and our interaction with it.

Google Desktop

Please note that throughout the rest of this section, ‘Google Desktop’ will be referred to as ‘GD’. In this section, we will describe the four components of the Google Desktop system. The first piece of GD is the central one, the one that might be called the GD core. It contains the search index. The second part is the GD crawler, which finds new files to index. The third is the collection of indexing components GD uses to process a file found by the crawler into a set of indices. The final part isn’t really a part of GD per sé, but is necessary for querying GD: a querying plugin. For now, let’s make sure the relationship between the crawler, the core, and indexing plugins is clear.

The Indexing Triangle

Indexing is the process GD uses make files and even more abstract data entities on any machine searchable. The starting point of this process is the crawler. The GD crawler is a background process of Google’s that roams the host computer’s hard drive for new or newly modified files. At this point, the crawler simply gives the GD core that file’s path along with information about what kind of file it is.

At this point, the ball is in the GD core’s court. The core has a number of indexing plugins, each registered to index a certain kind of file. So, when the core gets a notification from its crawler about a certain file, the core considers the file’s type and gives the file’s path to the indexing plugin that is registered for the file’s type.

At this point, it is the indexer’s job to “index” the file. Indexing though is not quite a direct process. Rather than calling some number of “indexing” methods on some core GD object, the indexer is given the means to create and send “events” to the GD core. An “event” is really a discretization of the descriptive information for a file. When GD receives these “events” from its indexers, GD processes them internally into new search indices.

At this point, it ought to be made clear that the indexing process for GD is really a triangle. The crawler finds a file that might give rise to new indices, so it communicates with the GD core.

What it communicates to GD's core is the file's path and file's type. Then, the GD core picks a number of indexers to index the file based on the file's type and the indexers' registered types, and sends the indexer the file's path and a means of creating events. The indexer then builds an event (although it can create more than one) from the descriptive information in the file, which it sends back to the core to become a new set of indices.

Registering COM Objects

It is easy to understand why GD indexing plugins are registered with GD. If not by registration, how else would GD know what kind of files an indexing plugin indexes? However, any code that queries GD also has to be registered. The reason here may be slightly more obscure, but the reason probably lies in a lack of understanding about what implementation form plugins take – whether they be indexing plugins or querying ones. Plugins to GD are not executables though they are binary. They are COM objects.

What are COM Objects?

COM objects are really pre-compiled classes. The same way a class has a public interface and hidden implementation, COM objects have visible interfaces and inaccessible implementations. In fact, the same way classes can be collected in libraries, COM objects can be collected in DLLs (dynamic linked libraries). But rather than a referenced class needing to be compiled with each compilation of the referencing project, a COM object already exists as binary somewhere within the Windows system and gets referenced and instantiated at runtime.

There are two aspects of COM objects that are entirely distinct from anything in compile-time encapsulations like classes. The first is globally unique identifiers – GUIDs. When referencing a class, the code knows where the class' code is because of compiled references to those pieces of code. For binary COM objects though, there needs to be another way of finding them; Microsoft's answer is this: assign every COM object a globally unique identifier, which will be used like the ISBN of a book to define a COM object's uniqueness as well as how the Windows runtime finds it. Therefore, if another already-compiled collection of classes/COM objects – such as Google's GD API – needs to call methods on another COM object, all it needs is that other COM object's GUID.

The second aspect of COM objects that irrevocably separates it from classes is the idea of registering a COM object with Windows. In order for Windows to ever find a COM object referenced in other code, Windows has to know the object's GUID. The details of this are entrenched in Windows OS programming methods, but remember the bookstore metaphor. In order for a bookstore to sell a book, it needs to be able to find it; and for that, it needs the book's ISBN. To register a COM object, one needs to run RegAsm.exe on its containing DLL. However, there is a second step akin to placing the book on the correct shelf, keeping with the bookstore metaphor. This step is placing the DLL in the Global Assembly Cache with the GACUtil.exe.

This is thus the minimum prerequisite knowledge to work with the GD API. This is because the GD API is entirely COM-based. All of its components are COM objects, and all of its plugins

are COM objects. For this reason, from this point on in the report, ‘COM object’ and ‘plugin’ will be synonymous; when the entity’s properties as a COM object are more relevant, ‘COM object’ will be used, and vice-versa.

The Google Desktop API

Ultimately, the GD API is used very differently for the indexer than for the querying plugin. In the beginning though, the process is nearly identical: the plugin must be registered for its duty with GD.

Registering a Plugin with GD

The GD API is easy to understand once things have been set in motion, but there are some perhaps unexpected features and requirements of the API, not the least of which is this first idea: registering the plugin. In fact, the need for this step is not entirely a technical one. In one step, the GD API’s registration mechanism provides the plugin’s GUID to GD, and ensures that the user intended to install the given plugin. The reason why this is not necessary in a purely technical sense is, querying GD through the API does not require GD to have knowledge of the querying plugin’s GUID. Nevertheless, when any plugin is registered with GD, a window pops up displaying the plugin’s description, which was passed to the registration method, to ask the user to verify that they wanted the given plugin to have access to their computer via GD.

In detail, there are essentially two registration steps. The first is the same for all plugins: The plugin’s GUID and description are provided to start registration. The second however is different for each kind of plugin. If the plugin is to be an indexer to GD, an `IGoogleDesktopRegisterIndexingPlugin` object is gotten from the `GoogleDesktopRegistrarClass` and used to register the plugin to handle files of a given extension. For querying plugins though, an `IGoogleDesktopRegisterQueryPlugin` is gotten from the same registrar class and used to get a “cookie” 32-bit integer, which has to be included in all GD API querying methods for the query to be successfully executed.

Implementing an Indexing Plugin

First, remember that COM objects are classes that are compiled. To implement an indexing plugin – which is a COM object, which is thus a class – the class that is the plugin must implement the `DgoogleDesktopFileNotify` interface. Implementing this interface means implementing its only method: the `HandleFile` method. So long as the class implements this method, and the compiled COM object is successfully registered as an indexer, you have an indexer. Because the at this point the method is empty, your indexer performs no indexing, but it will be invoked each time its registered file type is encountered by the crawler.

Implementing the `HandleFile` method really means two things, corresponding to using each of its two parameters. The first parameter is the full path to the file to be indexed. Using this parameter looks like opening the file and extracting from it descriptive, useful information. The second parameter is an “event factory.” An event factory creates instances of the Event objects the indexer uses to communicate to GD the relevant information in the given file. Therefore, using the event factory means creating the appropriate event (there are 12 kinds

ranging from generic “Indexable” events to “IM” events, representing individual instant messages), and populating it with the relevant information you extracted with the file’s path.

Implementing a Querying Plugin

Implementing a querying plugin centers around how you handle that “cookie” integer returned to the plugin class during its registration. The reason for this is that all of the querying methods of the GD API require the plugin’s registration cookie. This is a real problem because unless registration occurs before (and unregistration after) every bit of code in the COM object that needs to query GD, the cookie cannot possibly be stored in any of the classes members – static or otherwise. The reason this data disappears is because the COM object is created anew from the given binary every time other code references it.

A simplistic persistence solution would be to put the cookie in a read-only file somewhere on the hard drive. A standards-compliant solution on the other hand would put the cookie in a read-only XML file somewhere on the hard drive. Our solution in fact places the cookie in a certain registry key-value pair in the Windows registry. (The registry is an entire topic unto its own; suffice it to say the registry is used highly in legacy-style Windows development to store small bits of data that need to persist COM object lifetimes.) With the cookie then persisting, we can simply instantiate a `GoogleDesktopQueryAPIClass`, and call any of its Query methods with the saved cookie.

Referencing the GD API

An extremely important – and surprisingly non-trivial – aspect of using the GD API is referencing it in a .Net C# project. However, this topic is highly pragmatic and only distracts from the big picture. Suffice it to say, the API *can* be referenced in any .Net project. For more information than this though, please refer to Appendix I for a full, step-by-step exposition as to how.

Referencing the Querying Plugin in a C++ Program

The process for this is very similar to that of referencing the GD API in .Net. However, its process is also just as convoluted and distracting. In fact, referencing the querying plugin isn’t even directly related to the GD API because it is simply the process of referencing in C++ a COM object – a COM object that happens to have GD-related code in its binary. Again, it *can* be done, but please refer to Appendix K for the explicit how-to.

Developing the Plugins

We developed two plugins: a custom indexer for .JPG files, and a querying plugin that wrapped GD’s querying methods in a number of useful methods. We ran into various interesting – albeit frustrating – problems in getting things working. Ultimately, we succeeded in fully implementing the minimum functionality, but more work should be done to create full-featured indexing and querying.

Developing the Indexer

The indexer's role is to make all of our dataset's JPEGs searchable with their file names and all of their meta-data. As such, it must do two things: extract the meta-data from a JPEG given its path, and create and fill an appropriate event with the JPEG's meta-data in such a way that the JPEG will be found with a query incorporating any of its meta-data.

Actually, both of these processes are simple. To extract a JPEG's data, the file must be opened into an Image object (in .Net's System.Drawing namespace), which has a property, PropertyItems, which is a collection of all meta-data in bytes. Looping through and encoding each of these byte arrays into ASCII strings finds all of the JPEG's meta-data in text form. The next step is putting this text in the right place in an event object.

There were three event objects we might have used for the JPEGs appropriately: Indexable, File, or MediaFile events. The File event inherits the properties of Indexable, and the MediaFile inherits the properties of both File and Indexable. Among these, there were really two properties where the meta-data could be placed to create the desired indices: Content (from Indexable), and Keywords (from MediaFile).

Struggling with Tracing

In fact, there were many issues actually implementing the indexer. Most importantly, we couldn't be absolutely sure whether files were being found by the crawler and thus successfully indexed. We took two approaches to working through these questions: the first was tracing the indexing, and the second was searching on each piece of the given meta-data of JPEGs whose indexing was traced.

Tracing alone was harder and more questionable than all the other debugging tasks in developing the indexer (including [un]registration of the indexer as well as creating appropriately indexed events). Our first attempt and most reliable method was displaying pop-up windows with necessary trace information at appropriate steps in the indexer's HandleFile method. However, these dialogs required us to click OK before indexer execution could continue. For some reason, this interruption always caused the GD crawler to move on and not get back to another JPEG for about 5 minutes. Therefore, we opted for a much more tricky tracing approach: appending relevant information to a trace file.

As a COM object, the indexer can be invoked concurrently. What this really means is we would see a trace file created, but then there would be no data in it, and it would be "in use by another program" until GD was manually shut down. Our solution to this was a simple locking mechanism based on the IsReadOnly property of the file: busy looping until the file becomes writable, when we open the file for writing, thus locking it from all other instances of the indexer. However, this isn't a strict mutual exclusion method, and we continued to have cryptic moments when the file would be locked up in an instance of the indexer.

Struggling with Indexing

We were surprised in filling the appropriate event with meta-data to have no problems extracting the files' meta-data, but to instead have issues finding the appropriate event. A MediaFile is of course the appropriate event. However, there were always debugging problems when using a

MediaFile, and I could never figure out if it was the tracing failing or the MediaFile's Keywords property with the meta-data not being indexed sufficiently by GD. Nevertheless, tracing and querying was successful when indexing the JPEG as a File event with the meta-data parsed out and placed in the event's Content property.

Developing the Searcher

The querying plugin's role is to wrap GD's querying functionality. Beyond registering the plugin with GD, implementing the plugin consisted of deciding on a useful interface and implementing that interface. Designing the interface took a simple consideration of what the GD API required, versus what parameters we would actually change and what format we would want about the results we might need to use.

As for what the API requires, GD's querying methods are passed three things: the plugins registration cookie, the query string, and a category object representing what kind of events to search for, and a ranking object representing whether to order the results by relevance or newness. However, we would many times only want to specify a query string against all events, ordered by relevance.

As for the rest of our needs, we do not always want all of the results, or all of the information about the results. For instance, we want a method that returns only the count of all query results. We also want a method to return the execution time of the query. And while most of the time, we will only be specifying different query strings, searching against all events, ordering results by relevance. However, there will likely be situations wherein we would need to specify what categories to query and how to order our results, in addition to specifying the query string itself. Therefore, we arrived at six methods with two parameter sets against three return values. The two parameter sets were one specifying the query string, category, and ranking, and the other specifying only the query string. The three kinds of return values were a string representation of all results, the number of results, and the time it would take to execute the query in milliseconds.

Struggling with Return Values

Literally our only issue with implementing the querying plugin was how to pass the GD result set object directly to the caller. We would have liked to be able to deal directly with the IGoogleDesktopQueryResultSet object, rather than counting its results or concatenating its results into a string. However, for unresolved reasons, when we referenced the querying plugin in C++ when it had any GD API members in any return values of the public interface (and therefore likely, any parameters as well), we would have compilation errors due to the GD API members being unresolved identifiers. In the end, we just implemented each of the pieces of functionality we have gotten out of the result set object.

Google Desktop Plugin Development Summary

In the end, we did arrive at useful, functional plugins for indexing and querying. The indexer technically indexes the JPEGs as File events; however, their meta-data is searchable nonetheless. The querying plugin doesn't return actual IGoogleDesktopQueryResultSet objects, but it does return all the information we would want to get out of it at this point in development. By the

way, supplementing our querying plugin is as simple as adding a public method and recompiling.

As for future work, there are things that can be done to both the indexer and searcher that would aid us highly. For the indexer, it would be preferable to index the JPEGs as MediaFiles, not Files, because of the additional indices such as Width and Height this would entail. Also, it would be preferable to successfully add a thumbnail to the indexer's event as again, this would mean a substantially larger index. About the searcher, it would be preferable to directly wrap all of the GD API's querying methods, and allow referencing code to create its own wrappers for this fully presented functionality.

MPICH2

Introduction

In order for the eight computers to search their hard drives simultaneously, the idea of parallel computing had to be utilized. This is when the same task is executed simultaneously on several computers [12]. The goal is to achieve faster results. One large problem is broken into several smaller problems. As a result, the several smaller problems can be solved much faster than the one bigger problem. This is because the several smaller tasks can be solved all at once by the multiple computers instead of one computer solving the bigger problem all by itself [12]. This is an example of the “Single Program, Multiple Data” concept” where there are multiple computers executing the same program at independent points [13].

The computers need a way to talk to each other so they know what part of the problem to solve. They use a message passing interface (MPI), which is a programming library that allows computers in parallel to pass and receive messages between one another. MPI’s goals are productivity and portability [14]. This project is utilizing the specific, newer MPI library, MPICH2. Using the idea of parallelism, one problem is broken into smaller one’s called “processes.” These are distributed amongst the computers. This method is optimized when the shortest possible run time is achieved, and this occurs when each computer runs one process each [14].

Tools Required

MPICH2 is only one of the tools required to search the eight computers. The coding languages and the other software used also have to be taken into account.

Code Languages:

C++: MPICH2 is supported in either C++ or Fortran. For the use of this project we are using C++ because it is easier to merge with the other parts of the project.

Software:

MPICH2: This isn’t necessarily software, but as described above it is the programming library that allows the eight computers to send messages back and forth between each other. Specifically, for this project they are sending a message that says to perform a Google Desktop search. They are in turn sending back a message that contains their search results.

Microsoft Visual Studio 2005 Express Edition: This is used to write the program using MPICH2. It will call the custom Google Desktop search. Once the program is successfully written, the executable file will be placed on all eight computers in the TEMP folder on the C:\ drive. More on this will be explained later.

Program Development

Now that the programs and the tools have been laid out, the program can be developed. Before the coding could start, however, the development environment had to be set up. After everything was set up, a provided example MPI program was run to learn how a message passing interface worked and how to use the command line arguments. Following that, the actual development started. Next, it had to be determined how the host computer would tell the other computers to search. Either an HTTP request could be sent to each computer or each computer would be told to instantiate and call a Google Desktop Query object. Both methods have been visited and tested, and it was decided to use a Google Desktop Query object. After this decision was made, a mock MPI program was created as a “template” for the final program. The final step was to actually integrate Google Desktop.

Setup

However, before any of that could be completed, the development environment had to be prepared. The following is just a brief outline of everything that had to be set up. It does not go into technical specifics. The complete technical set up can be referenced in *Appendix L, MPICH2 Set up*. First, the appropriate library, executable and include files for the MPICH2 library had to be added to the directories in Microsoft Visual Studio. Next, the Microsoft Software Development Kit (Platform 2) library, executable, and include files had to be added to the same directories. After that, the appropriate linker libraries for MPICH2 had to be added. These were “mpi.lib” and “cxx.lib.” After these steps were completed, Microsoft Visual Studio was completely set up. However, the overall set up isn’t complete.

There were some general Microsoft environment variables that had to be changed. For example, MPICH2 programs have to be compiled from the command line. In order to do this, mpirun.exe had to be included in the Paths under Properties and My Computer. This is accomplished by adding the path to the MPICH2 bin folder. The location to the TEMP folder also needs to be added in this same location. This is because all the executable files for the running program are going to be put in the TEMP folder, and this tells the computer where to look for them.

When all of these tasks were completed, the team tried to compile and run the test program. However, it wouldn’t work. The program wouldn’t even load. We thought that there was something wrong with the set up on our end. We then realized that the firewalls on each of the computers had to be turned off. This was something that is an administrative task. However, this still didn’t solve the problem. Now the program would load, but the IP addresses for each of the machines couldn’t be called. We finally asked someone from the computing staff, and it was determined that it was a school network-related error. The error did actually occur when the IP addresses were called. The machines themselves saw the name of their computer as “computername.adit.mines.edu,” but when the IP address request was sent to the school server, which is Linux based, it saw the names of the machines as “computername.mines.edu.” The Linux server said that these computers didn’t exist and sent the IP request to the Windows DNS. However, none of the computers are registered there. Therefore none of the IP addresses could be called. This problem was fixed by having each of the machines call their IP addresses dynamically. After this fix, the development environment was finally set up.

Example Program

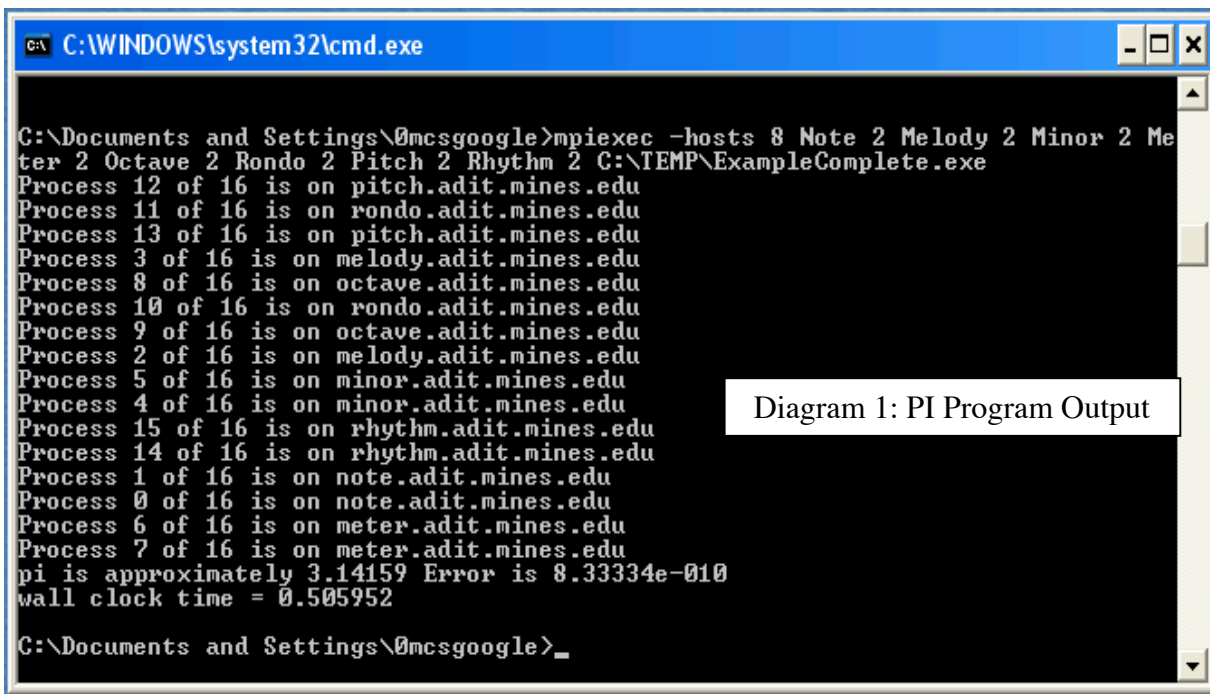
After all of the set up was complete and the set up problems were solved, we got the example program to run. It was provided to us by the people who developed MPICH2, and it came in the MPICH2 program folder. It is a program that calculates the number PI using a Taylor series. The program was compiled on one computer, and then the executable was placed in the TEMP folder on each of the eight computers under the same name, "Example.exe."

Once this was completed, the program could be built. Since the executable file is on all of the computers it doesn't matter which of them it is ran on. To run the program, pull up the command prompt by going to Start->Run->cmd. Then on the command line type the following:

```
"mpiexec -hosts 8 melody 2 minor 2 note 2 rondo 2 meter 2 octave 2 rhythm 2 pitch 2  
C:\TEMP\Example.exe"
```

The breakdown of this command can be seen in *Appendix M*.

The output from this program displays the number of the process and on what computer that process number was preformed. After all the process numbers are printed, it displays calculated value of PI and the error. It also displays the time that it took to complete the task. The output of this program for each computer running two processes is in *Diagram 1*. The complete code can be found in *Appendix N: MPICH2 Example Code*.



```
C:\WINDOWS\system32\cmd.exe  
  
C:\Documents and Settings\0mcsgoogle>mpiexec -hosts 8 Note 2 Melody 2 Minor 2 Me  
ter 2 Octave 2 Rondo 2 Pitch 2 Rhythm 2 C:\TEMP\ExampleComplete.exe  
Process 12 of 16 is on pitch.adit.mines.edu  
Process 11 of 16 is on rondo.adit.mines.edu  
Process 13 of 16 is on pitch.adit.mines.edu  
Process 3 of 16 is on melody.adit.mines.edu  
Process 8 of 16 is on octave.adit.mines.edu  
Process 10 of 16 is on rondo.adit.mines.edu  
Process 9 of 16 is on octave.adit.mines.edu  
Process 2 of 16 is on melody.adit.mines.edu  
Process 5 of 16 is on minor.adit.mines.edu  
Process 4 of 16 is on minor.adit.mines.edu  
Process 15 of 16 is on rhythm.adit.mines.edu  
Process 14 of 16 is on rhythm.adit.mines.edu  
Process 1 of 16 is on note.adit.mines.edu  
Process 0 of 16 is on note.adit.mines.edu  
Process 6 of 16 is on meter.adit.mines.edu  
Process 7 of 16 is on meter.adit.mines.edu  
pi is approximately 3.14159 Error is 8.33334e-010  
wall clock time = 0.505952  
  
C:\Documents and Settings\0mcsgoogle>_
```

Diagram 1: PI Program Output

I ran this program several times, and experimented with the different number of process to see if the error changed and the change in the execution time. I started out with one executed one computer. This was the shortest execution time; however, it is a very unrealistic setup because information is more likely to be spread out amongst several computers. After this I added

computers with one process per computer. The execution time increased up until there were eight computers and eight processes. This was our maximum computer number, and, therefore, only the process number could be increased after this point. This means that more than one process is run per computer. The execution time increased at this point. Therefore, when a minimum of one process is run per computer, maximization occurs when only one process is run per computer.

I thought that there would be some kind of relationship between the number of computers/processes and the error calculation of PI. However, from what I could tell from my limited testing, the error was quite random, and it did not follow any pattern. The results from my limited testing can be seen in the *Table 1*.

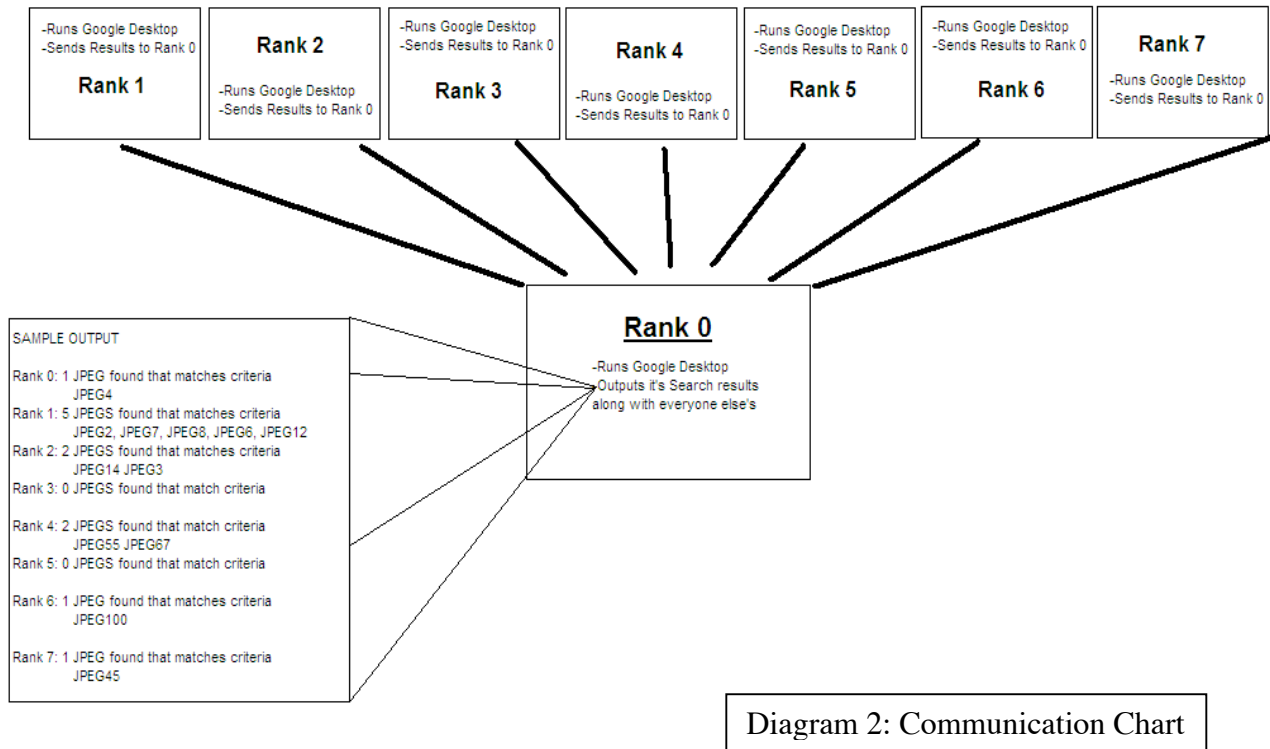
# of Computers	# of Processes	Execution Time	Error in PI
1	1	.00113789	8.33341e-010
2	2	.00263713	8.33339e-010
3	3	.00576285	8.33339e-010
4	4	.0044770075	8.33331e-010
5	5	.00617744	8.3333e-010
6	6	.157334	8.33331e-010
7	7	.159299	8.33331e-010
8	8	.116259	8.33332e-010
8	16	.472489	8.33334e-010
8	24	1.38179	8.33335e-010

Table 1: MPICH2 Example Tests

Mock Setup Program

After MPICH2 was running and there was a basic understanding of how a message passing interface worked, the development of the program could begin. The program that had to be written for this project has to tell to the eight computers to perform a Google Desktop search for some specific metadata. The eight computers then have to send back a message that contains their search results. The results then have to be printed out in an organized form. It was unclear for a while during the Google Desktop development how the message would be sent and how search results would be sent back.

In the meantime, however, it was thought best to develop a mock set up program. We thought this would at least provide some type of skeleton for when we would know how the messages would be sent and what would be received. Furthermore, the specific lines of code would just need to be filled in when the when the Google Desktop plug in was completed. Before this



This mock set up program uses a simple class that was created just for this purpose. The class holds simple string and integer information that is comparable to the returned information from a search such as the number of items found and the name of the items. The set up program has each rank outputting the information in an organized fashion similar to the way we would want them to output the information at the completion of the search. It says what computer is in charge of the information; this is supposed to be similar to saying where the returned objects are found. Each rank also outputs a phone number; this is similar to returning the total number of JPEGs found during the search on each computer. Finally, each rank outputs some basic string information in the form of a first and last name; this is similar to each rank returning the names of the JPEGs found. The complete code can be found in *Appendix C*. The example output can be found below in *Diagram 3*.


```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\0mcs.google>mpiexec -hosts 2 octave 4 rondo 4 C:\TEMP\MockSetup2.exe
I am rank 7 of 8
This is my information:
Mr Hyde 9874563
I am rank 4 of 8
I am rank 0 of 8
This is my information:
bailey kelly 1234567
This is my information:
justin cirbo 5678945
I am rank 6 of 8
This is my information:
Dr Jekel 7894561
I am rank 5 of 8
This is my information:
Mr Bean 6789123
I am rank 1 of 8
This is my information:
olen davis 2345678
I am rank 2 of 8
I am rank 3 of 8
This is my information:
kavi macklin 3456789
This is my information:
shaun fishbein 4567890

C:\Documents and Settings\0mcs.google>

```

Diagram 3: Mock Setup Output

HTTP Request Approach

Although Google outright stated that the HTTP method is out of date, we decided to try this direction for simplicity's sake [15]. It was thought that this method would require less lines of code within the MPI calls, it would be easier to parse out the results in Extensible Markup Language (XML) format, and it would be easier than trying to program with COM objects natively within C++. Furthermore, how it would work is within the MPI code, each node would make an HTTP request on the local server and the local port to Google Desktop. That tells each computer to do a Google Desktop search. The results are then put into an XML page and sent back to the host computer. The host computer then parses out the XML page and displays the search results.

The example code can be seen in *Appendix P* [16]. The code itself was found on multiple open source sites. It creates a function called *request* that takes four parameters:

- 1) char* host name: This is the name of the host either in form of the IP address or in www.hostname.com form. It needs to be in quotes when the function is called.
- 2) char* api: This is the additional direction information after the host name or the IP address. Its in the form of `"/.../.../.../"`. It has to be in quotes.
- 3) char* parameters: This is not applicable to our use. Send it a blank string in the form of `""`.
- 4) string& message: This is note applicable to our use. Send it a blank string in the form of `""`.

When you connect to just a random website on the internet, you use the standard port 80. This is set within the function definition. For example, to bring up the webpage <http://www.golfsurround.com/usopen/2007/scoring/index2.html>, you would use the following function call:

```
request("www.golfsurround.com", "/usopen/2007/scoring/index2.html/", "", "")
```

The output would then bring this page up in html format in the command prompt screen. We used examples like this to test this code. Web pages with extensive graphics and videos would just pull up a redirect link within the command prompt because it couldn't parse out the advanced html.

To do a call to run a Google Desktop search, a call has to be made to unique URL for each computer [15]. This URL was retrieved through a few lines of simple C# code that was written specifically for this purpose. This code can be seen in *Appendix Q*. The complete list of unique URLs for each computer can be seen in *Appendix R*. The URL for my personal computer is the following:

<http://127.0.0.1:4664/search&s=-FsFyw84DzMIXvzZtnLXaepMy3E?q=>

The <http://127.0.0.1:4664/> is the same for all the computers. The 127.0.0.1 is the local IP address and the 4664 is the local port. The rest of the URL is the unique search query for each computer. The string you want to search for is added after the last equal sign along with the correct parameters to put the search results' in XML format. For example, if you want to do a Google Desktop search for the word "Google Desktop" you would type the following into your internet browser: <http://127.0.0.1:4664/search&s=-FsFyw84DzMIXvzZtnLXaepMy3E?q=Google+Desktop&format=xml>.

The problems with the HTTP request method of calling a Google Desktop search began when we tried to put this URL into the code. The first thing that had to be done was the port had to be changed from 80 to 4664. This is done within the function definition. Just simply replace '80' with '4664.' The function call looks like the following:

```
request("127.0.0.1," "/search&s=-  
FsFyw84DzMIXvzZtnLXaepMy3E?q=Google+Desktop&format=xml," "", "");
```

When we ran the program with this function call, it didn't complete the search. The page that was brought up in the command prompt was a Google Desktop page that said your request could not be completed and that we had reached this page on accident. The page was in HTML format, and not the XML format like requested. The encouraging thing about this error page was that it was from Google Desktop. This means that the request connected to the correct port and the correct IP address. The fact that the page wasn't in XML format and it didn't complete the search meant that we thought the program wasn't passing in the correct additional string. We tried four days worth of debugging with this program, but we could not get it to work.

COM Object Approach

At this point we decided to try the COM object method of calling a Google Desktop search. All of the search implementations are within the Google Desktop API. Therefore, all of these functions are within the Google Desktop side of the development. Furthermore, a COM object was developed, as discussed earlier, to for the search requests. All that has to be done on the MPI side of this is instantiate and implement the COM object within the MPICH2 code.

While the Google Desktop JPEG indexer was being developed, we wanted to try and run practice searches on the eight computers. These practice searches would just be a normal desktop search

of the computer. The purpose of running these searches is to make sure that the Google Desktop environment could be ran in parallel. This still required some development from the Google Desktop side because it required the use of the ParallelSearchComponents.dll and the GoogleDesktopAPIStrong.dll.

The actual programming of this demo implementation was quite simple. We took the Mock Set Up program and removed the calls to the objects that were created. These were replaced by a call to a function called Query() that contained the instantiation and implementation if the Google Desktop Query API. The complete code can be seen in *Appendix S*.

Once the appropriate .dll's were registered for the Google Desktop objects, the program could be compiled and ran. To run this program in parallel, however, the .dll files had to installed and registered on all eight machines. Also, the executable file had to be transferred and put into the TEMP folder. A similar call on the command line like the one used in the original demo program was used to run the program. This is where our project hit its final snag. All though the program compiled fine in Microsoft Visual studio, we got a run time error when the program started. The exact error message can be seen in the screen shot in *Diagram 4*.

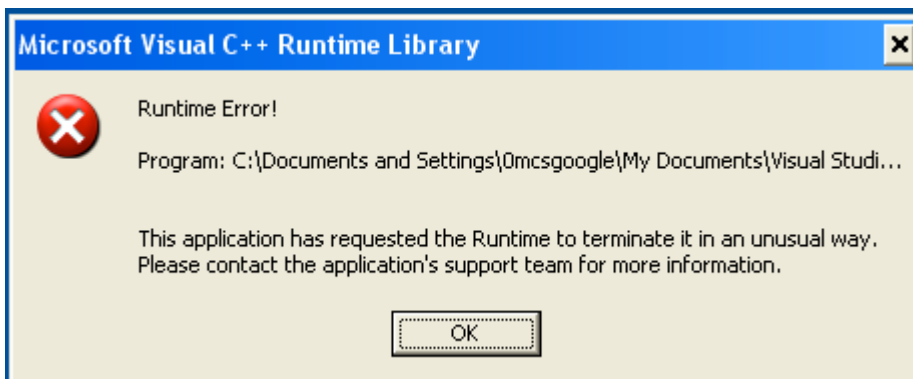


Diagram 4: Final Error Message

According to the technical department, this error is most likely due to .NET compatibility. Apparently, MPICH2 (even more generally MPI) is not compatible with .NET versions of 2.0 and later. It is, however, compatible with .NET 1.1. We received this error on the very last day we had for implementation. Therefore, we were unable to debug this error.

Conclusions and Future Work

Although the MPICH2 part of the project wasn't fully completed, there were still several things that were accomplished. We were able to configure an MPI setup for Windows and Microsoft Visual Studio C++. We also learned about the MPICH2 environment through the exploration of the example PI program. Most importantly, the framework for the MPI/Google Desktop program has been written. Actually, the entire MPICH2/Google Desktop program has been written, and it compiles perfectly fine. The error that we received during runtime has to do with the environment that we are working and the versioning of the software that we used to compile the plug-ins.

In order for this project to continue in its entirety, obviously, the .NET bug has to be fixed. Once this is completed, the actual searches can be run. Even though the code that is currently in the MPI/GD integration program only runs a normal GD search sans JPEGs, nothing needs to be changed in order to actually search for JPEGs. In order to run the metadata searches, changes need to be made on the plug-in side of the implementation. Therefore, what was produced during this field session is what is needed to perform the parallel searches in full.

Conclusion

Although we do not have search results with which to analyze the effectiveness of parallel searching and indexed searching, we have been able to produce two Google Desktop plug-ins for indexing and querying that can be utilized for these searches. We have also created a dataset with searchable metadata and a process for generating files for a dataset. Finally we have developed an MPICH2 utility that is capable of using the Google Desktop plug-ins for parallel searching.

Future Work

The original scope of this project actually included performing and analyzing the parallel searches of the metadata. We were unable to get to this portion of the project due to the constraints on time. To fully complete this project, we want to combine parallelism using MPI with efficient search techniques using Google Desktop and compare the results with those found using Windows' old file search. The efficiency of these data searches will be numerically analyzed and represented graphically.

Originally, we had four different searches that we wanted to run:

- 1) A one computer serial search of the entire dataset. The dataset will not be indexed for this search. To be able to search the entire dataset, we would use an external hard drive.
- 2) A multiple computer serial search with the data set distributed amongst the computers. The data set will not be indexed for this search.
- 3) A one computer indexed search of the entire dataset. This is similar to the first search except that Google Desktop will index the data before it is searched.
- 4) A multiple computer indexed search.

Throughout the project, we also had some other thoughts of different ways to analyze the data and other searches we could perform:

- 1) Change the amount of data that each computer has. For example, we would run the multiple computer searches with each computer containing equal amounts of data. Then we would try running the searches with the data unevenly distributed amongst the computers. The purpose of this would be to consider an optimum file organization method for companies with large amounts of searchable data.
- 2) Determine if the Law of Diminishing Marginal Returns holds for parallel computing. We would like to see if there are a maximum number of computers that creates peak

searching performance, where after this number each additional computer diminishes performance of the searching task.

- 3) See if we can predict search times for other numbers of computers. We have eight computers. We can run searches on one computer, then two computers, etc up to eight computers. We will then have search times for eight computers, and we want to see if we can interpolate these results to more computers.

References

- [1] Elysium Ltd, “JPEG,” <http://www.jpeg.org/> Accessed June 20, 2007.
- [2] “EXIF.org,” <http://www.exif.org/> Accessed June 20, 2007.
- [3] “e695. Saving a Generated Graphic to a PNG or JPEG File,” <http://www.exampledepot.com/egs/javax.imageio/Graphic2File.html> Accessed June 20, 2007.
- [4] MSDN “How to: Read Image Metadata” <http://msdn2.microsoft.com/en-us/library/xddt0dz7.aspx> Accessed June 20, 2007.
- [5] “Description of Exif File Format,” <http://www.media.mit.edu/pia/Research/deepview/exif.html> Accessed June 20, 2007.
- [6] “Normal distribution,” http://en.wikipedia.org/wiki/Normal_distribution Accessed June 20, 2007.
- [7] “Generating Gaussian Random Numbers,” <http://www.taygeta.com/random/gaussian.html> Accessed June 20, 2007.
- [8] “Using the LockBits method to access image data,” <http://www.bobpowell.net/lockingbits.htm> Accessed June 20, 2007.
- [9] “Image File Builder” <http://www.codeproject.com/useritems/ImageFileBuilder.asp> Accessed June 20, 2007.
- [10] “JPEG Metadata Format Specification and Usage Notes,” http://java.sun.com/j2se/1.5.0/docs/api/javax.imageio/metadata/doc-files/jpeg_metadata.html Accessed June 20, 2007.
- “Parallel Computing”
- [12] http://en.wikipedia.org/wiki/Parallel_computing Accessed June 15, 2007
- “Single Program Multiple Data”
- [13] http://en.wikipedia.org/wiki/Single_program_multiple_data, Accessed June 15, 2007
- “Message Passing Interface”
- [14] http://en.wikipedia.org/wiki/Message_passing_interface, Accessed June 15, 2007
- “Google Desktop Query API”
- [15] <http://desktop.google.com/dev/queryapi.html>, Accessed June 15, 2007

“HTTP Source Code”

[16] http://www.gamedev.net/community/forums/topic.asp?topic_id=324756, Accessed June 15, 2007

Appendix A: Dataset Size per Computer

Dataset Size Per Computer	
Computer	Data (GB)
Rondo	9.46
Rhythm	9.47
Pitch	9.52
Octave	9.49
Note	9.49
Minor	9.48
Meter	9.51
Melody	9.49

Appendix B: Picture Generator

Pictures Class:

//Picture Generator

//Sources: <http://www.exampledepot.com/egs/javax.imageio/Graphic2File.html>

<http://schmidt.devlib.org/java/image-faq/read-write-image-files.html>

<http://forum.java.sun.com/thread.jspa?threadID=478634&start=0&tstart=0>

```
import java.awt.Color;
import java.awt.GradientPaint;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.awt.image.RenderedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
```

```
public class Pictures extends JFrame {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    private JTextField text;

    private JPanel panel;

    /*
     * Default Constructor for Pictures Object Generates JPEGs
     */
}
```



```

*/
public Pictures(int numberpictures, int startingpt, String location,
               String name, String extension) {

    // instantiate frame, panel and textfield
    super();
    panel = new JPanel();
    text = new JTextField();

    // configure frame
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setSize(100, 100);

    // used to calculate total runtime
    long starttime = System.currentTimeMillis();

    // loop that controls how many JPEGs are generated
    for (int i = 0; i < numberpictures; i++) {

        // Random object used to generate random numbers for JPEG //specs
        Random rand = new Random();

        // height and width for the BufferedImage (translated later //to JPEG)
        // height and width are random values between 2000 and 3000
        int width = 2000 + rand.nextInt(1000);
        int height = 2000 + rand.nextInt(1000);

        // Create a buffered image in which to draw
        BufferedImage bufferedImage = new BufferedImage(width,
                                                         height, BufferedImage.TYPE_4BYTE_ABGR_PRE);

        // Create a graphics contents on the buffered image
        Graphics2D g2d = bufferedImage.createGraphics();

        // Draw graphics
        // Graphics are drawn based on random boolean types
        // Graphics have random colors, sizes, and placements
        g2d.setColor(new Color(rand.nextInt(255),
                               rand.nextInt(255), rand.nextInt(255)));
        g2d.fillRect(0, 0, width, height);
        if (rand.nextBoolean() == true) {
            // Draws random oval
            g2d.setColor(new Color(rand.nextInt(255),
                                   rand.nextInt(255), rand.nextInt(255)));
            g2d.fillOval(rand.nextInt(300), rand.nextInt(300), width - rand.nextInt(3000),
                        height - rand.nextInt(3000));
        }
        if (rand.nextBoolean() == true) {
            // Draws another random oval
            g2d.setColor(new Color(rand.nextInt(255),
                                   rand.nextInt(255), rand.nextInt(255)));
            g2d.fillOval(rand.nextInt(300), rand.nextInt(300), rand.nextInt(3000),
                        rand.nextInt(3000));
        } else {

```

```

// Draws a random line with Gradient coloring if
// above oval not
// drawn
int x = rand.nextInt(3000);
Line2D ln = new Line2D.Double(x,rand.nextInt(2000), x, rand.nextInt(2000));

GradientPaint gp = new GradientPaint(75, 75,
    new Color(rand.nextInt(255), rand.nextInt(255), rand.nextInt(255)),95,
    95,
    new Color(rand.nextInt(255), rand.nextInt(255),
    rand.nextInt(255)), true);

// Fill with a gradient.
g2d.setPaint(gp);
g2d.fill(ln);
}
if (rand.nextBoolean() == true) {
// Draws random Rectangle
g2d.setColor(new Color(rand.nextInt(255),
rand.nextInt(255), rand.nextInt(255)));
g2d.fillRect(rand.nextInt(200), rand.nextInt(200), width - rand.nextInt(3000),
height - rand.nextInt(3000));
} else {
// Draws random ellipse with Gradient coloring if
// above rectangle isn't drawn
Ellipse2D elp = new
    Ellipse2D.Double(rand.nextInt(3000),
    rand.nextInt(3000), rand.nextInt(3000));

GradientPaint gp = new GradientPaint(75, 75,
    new Color(rand.nextInt(255), rand.nextInt(255), rand.nextInt(255)),95,
    95,
    new Color(rand.nextInt(255), rand.nextInt(255),
    rand.nextInt(255)), true);

// Fill with a gradient.
g2d.setPaint(gp);
g2d.fill(elp);
}
if (rand.nextBoolean() == true) {
// Draws another random rectangle
Rectangle2D rect = new
    Rectangle2D.Double(rand.nextInt(3000),
    rand.nextInt(3000), rand.nextInt(3000),
    rand.nextInt(3000));

GradientPaint gp = new GradientPaint(75, 75,
    new Color(rand.nextInt(255), rand.nextInt(255), rand.nextInt(255)),95,
    95,
    new Color(rand.nextInt(255), rand.nextInt(255),
    rand.nextInt(255)), true);

// Fill with a gradient.
g2d.setPaint(gp);
g2d.fill(rect);
} else {

```

```

        // Draws another random rectangle if above rectangle //isn't drawn
        g2d.setColor(new Color(rand.nextInt(255),
            rand.nextInt(255), rand.nextInt(255)));
        g2d.fillRect(rand.nextInt(200), rand.nextInt(200),
            rand.nextInt(3000), rand.nextInt(3000));
    }

    // Graphics context no longer needed so dispose it
    g2d.dispose();

    // Create an image to save
    RenderedImage rendImage = bufferedImage;

    // Write generated image to a file
    try {

        // Save as JPEG using the ImageIO to write a jpg //formatted file
        File file = new File(location + "/" + name + (
            startingpt + i) + "." + extension );
        ImageIO.write(rendImage, "jpg", file);

    } catch (IOException e) {
    } catch (Exception e) {

        e.printStackTrace();
    }
}
long endtime = System.currentTimeMillis();

// Display name of image to panel
text.setText("Time: " + (endtime - starttime));
panel.add(text);
getContentPane().add(panel);
setVisible(true);
System.out.println("Time: " + (endtime - starttime));
}
}

Setup Class (GUI);
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class Setup extends JFrame implements ActionListener {

    private JPanel pane;

    private JTextField numpictures;

    private JTextField firstpicture;

```

```

private JTextField folderlocation;

private JTextField extensiontxt;

private JTextField picturenames;

private JButton enter;

private JLabel numpictlbl;

private JLabel fstpiclbl;

private JLabel fldloclbl;

private JLabel picnamelbl;

private JLabel extlbl;

Setup() {

    // Initial setup of JFrame
    super("Setup");
    this.setSize(1200, 150);

    // Setup of JPanel
    pane = new JPanel();

    // Initializing labels, textfields and button
    numpictlbl = new JLabel("Enter number pictures to be created: ");
    fstpiclbl = new JLabel("Enter starting number of pictures to be
        created: ");
    fldloclbl = new JLabel("Enter location for pictures to be placed:
        ");
    picnamelbl = new JLabel("Enter the name associated with pictures:
        ");
    extlbl = new JLabel("Enter the extension associated with
        pictures: ");

    numpictures = new JTextField(15);
    firstpicture = new JTextField(15);
    folderlocation = new JTextField(100);
    picturenames = new JTextField(60);
    extensiontxt = new JTextField(10);

    enter = new JButton("Enter");
    enter.addActionListener(this);

    // Add objects to pane
    pane.add(numpictlbl);
    pane.add(numpictures);
    pane.add(fstpiclbl);
    pane.add(firstpicture);
    pane.add(fldloclbl);
    pane.add(folderlocation);
    pane.add(picnamelbl);
    pane.add(picturenames);

```

```

pane.add(extlbl);
pane.add(extensiontxt);
pane.add(enter);

// Final setup of JFrm
this.getContentPane().add(pane);
this.setVisible(true);
}

public static void main(String[] args) {
    // create Setup object
    new Setup();
}

public void actionPerformed(ActionEvent arg0) {
    // TODO Auto-generated method stub
    int numberpictures, start, height, width;
    String location, name, extension;

    numberpictures = new Integer(numpictures.getText());
    start = new Integer(firstpicture.getText());
    //location = "testimages/testjavaimages";
    location = folderlocation.getText().toString();
    name = picturenames.getText().toString();
    extension = extensiontxt.getText().toString();

    //Create Pictures Object
    new Pictures(numberpictures, start, location, name, extension);
}
}
}

```

Appendix C: Tag Adder Versions

```

//Basic Tag Adder Program
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Drawing.Imaging;
using System.Timers;
namespace ConsoleApplication1
{
    class ProvideJpeg
    {
        private Random rand;
        static void Main(string[] args)
        {
            ProvideJpeg peg = new ProvideJpeg();
            peg.rand = new Random();
            DateTime beginningSystemTime = DateTime.Now;
            for (int i = 0; i < 2400; i++)
            {
                Bitmap jpeg = new Bitmap(@"C:\Documents and Settings\0mcs.google\workspace\Picture

```

```

        Generator\images4>taggedjavaimages" + (26401 + i) + ".jpg");
jpeg.SetPropertyItem(peg.NewTag(jpeg));
jpeg.SetPropertyItem(peg.NewTag2(jpeg));
jpeg.SetPropertyItem(peg.NewTag3(jpeg));
jpeg.SetPropertyItem(peg.NewTag3(jpeg));
jpeg.Save(@"C:\Documents and Settings\0mcsgoogle\My Documents\Parallel Search\Normaltest0xffff7,1-
2400\normaltestimages" + (1 + i) + ".jpg");
    }
    DateTime endingSystemTime = DateTime.Now;
    Console.WriteLine(endingSystemTime - begginingSystemTime);
    Console.ReadLine();
}
private PropertyItem NewTag(Bitmap bit)
{ ... }
}
}

```

Possible NewTag(...) functions:

Random Byte Tag

```

private PropertyItem NewTag(Bitmap bit)
{
    // Get PropertyItem contained in all the JPEGs
    PropertyItem prop2 = bit.GetPropertyItem(20624);
    // Set the id, type, value and length for the tag to be added
    prop2.Id = 271;
    prop2.Type = 2;
    byte[] y = { 0, 0, 0, 0 };
    // Fills byte[] with random byte values
    rand.NextBytes(y);
    prop2.Value = y;
    prop2.Len = y.GetLength(0);
    // Return the PropertyItem with the new characteristics
    return prop2;
}

```

Constant Tag

```

private PropertyItem NewTag(Bitmap bit)
{
    PropertyItem prop2 = bit.GetPropertyItem(20624);
    prop2.Id = 0xff55;
    prop2.Type = 2;
    // Value to be input into tag
    byte[] y = { 0x4C, 0x41, 0x4E, 0x4C, 0x2C, 0x20, 0x50, 0x53 };
    prop2.Value = y;
    prop2.Len = y.GetLength(0);

    return prop2;
}

```

Weighted Tag (10%, 15%, 25%,50%)

```

private PropertyItem NewTag(Bitmap bit)
{
    PropertyItem prop2 = bit.GetPropertyItem(20624);
    prop2.Id = 0xffff6;
    prop2.Type = 2;
    if (rand.NextDouble() <= .1)
    {
        byte[] y = { 48, 49, 50, 51 };
        prop2.Value = y;
    }
}

```

```

    prop2.Len = y.GetLength(0);
}
else if (rand.NextDouble() <= .25)
{
    byte[] y = { 65, 66, 67, 68, 69 };
    prop2.Value = y;
    prop2.Len = y.GetLength(0);
}
else if (rand.NextDouble() <= .5)
{
    byte[] y = { 88, 89, 90 };
    prop2.Value = y;
    prop2.Len = y.GetLength(0);
}
else
{
    byte[] y = { 97, 98, 99, 100, 101, 102 };
    prop2.Value = y;
    prop2.Len = y.GetLength(0);
}
return prop2;
}

```

Box-Muller Tag

```

private PropertyItem NewTag(Bitmap bit)
{
    PropertyItem prop2 = bit.GetPropertyItem(20624);
    prop2.Id = 0xfff7;
    prop2.Type = 2;
    byte[] y = { 0, 0, 0, 0 };
    // Gaussian() function uses Box-Muller Transformation described below
    int x = (int)(Gaussian() * 100);
    System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding();
    string value = x.ToString();
    y = encoding.GetBytes(value.ToCharArray());
    prop2.Value = y;
    prop2.Len = y.GetLength(0);
    return prop2;
}

```

Gaussian

//supposed to generate random numbers, originally in pairs

```

private float Gaussian()
{
    float x1, x2, w, y1; //y2; - second value which algorithm usually produces
    do
    {
        x1 = (float)(2.0 * rand.NextDouble() - 1.0);
        x2 = (float)(2.0 * rand.NextDouble() - 1.0);
        w = x1 * x1 + x2 * x2;
    } while (w >= 1.0);
    w = (float)(Math.Sqrt((-2.0 * Math.Log(w, Math.E)) / w));
    y1 = x1 * w;
    //y2 = x2 * w;
    return y1;
}

```

Uniform Discrete (25% each of 4 values)

```

private PropertyItem NewTag(Bitmap bit)
{
    PropertyItem prop2 = bit.GetPropertyItem(20624);
    prop2.Id = 271;
    prop2.Type = 2;
    if (rand.NextDouble() < .25)
    {
        prop2.Len = 4;
        byte[] y = { 0x78, 0x79, 0x7A };
        prop2.Value = y;
    }
    else if (rand.NextDouble() < .5)
    {
        prop2.Len = 6;
        byte[] y = { 0x41, 0x42, 0x43, 0x44, 0x45 };
        prop2.Value = y;
    }
    else if (rand.NextDouble() < .75)
    {
        prop2.Len = 5;
        byte[] y = { 0x53, 0x4F, 0x4E, 0x59 };
        prop2.Value = y;
    }
    else
    {
        prop2.Len = 6;
        byte[] y = { 0x43, 0x61, 0x6E, 0x6F, 0x6E };
        prop2.Value = y;
    }
    return prop2;
}

```

Discrete Normal (2.3%, 13.6%, 34.1%)

```

private PropertyItem NewTag3(Bitmap bit)
{
    PropertyItem prop2 = bit.GetPropertyItem(20624);
    prop2.Id = 0xfff9;
    prop2.Len = 4;
    prop2.Type = 2;
    if (rand.NextDouble() < .023)
    {
        byte[] y = { 65, 66, 67 };
        prop2.Value = y;
    }
    else if (rand.NextDouble() < .159)
    {
        byte[] y = { 68, 69, 70 };
        prop2.Value = y;
    }
    else if (rand.NextDouble() < .5)
    {
        byte[] y = { 97, 98, 99 };
        prop2.Value = y;
    }
    else if (rand.NextDouble() < .841)
    {

```



```

        byte[] y = { 100, 101, 102 };
        prop2.Value = y;
    }
    else if (rand.NextDouble() < .977)
    {
        byte[] y = { 49, 50, 51 };
        prop2.Value = y;
    }
    else
    {
        byte[] y = { 120, 121, 122 };
        prop2.Value = y;
    }

    return prop2;
}

```

Appendix D: Parallel Search Tag Adder (Saves to new folder and deletes old folder)

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Text;
using System.Windows.Forms;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        private Random rand;

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            rand = new Random();
            // Get all pictures in Directory
            DirectoryInfo directory = new
                DirectoryInfo(this.directuntagged.Text);
            DateTime beginningSystemTime = DateTime.Now;
            // Create a FileInfo[] from the directories
            FileInfo[] files = directory.GetFiles("*" +
                this.initexttxt.Text);

```

```

int i = 0;

// Loop through FileInfos
foreach (FileInfo file in files)
{
    // Create filestream which can be closed
    // so that files can be deleted
    FileStream filestm = file.Open(System.IO.FileMode.Open);
    // Create Bitmap from filestream
    Bitmap bit = new Bitmap(filestm);

    bit.SetPropertyItem(this.CameraMake(bit));
    ...

    // Number assigned to new file
    int start = (int)(startnumber.Value) + i;

    // Save the file
    bit.Save(this.directuntagged.Text + @"\csmlanlps" + start +
            this.extfintxt.Text);

    // Close filestream
    filestm.Close();
    // Delete File
    file.Delete();

    i++;
}
DateTime endingSystemTime = DateTime.Now;

this.updatetxt.Text = (endingSystemTime -
        begginingSystemTime).ToString();
}

    // Various tags to be added
    ...

}

}

```

Appendix E: Parsing Code

<http://www.codeproject.com/useritems/ImageFileBuilder.asp>

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Text;
using System.Windows.Forms;

```

```

namespace ParallelSearchJPEGs
{
    public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();
        }

        private void uxJPEGBrowseButton_Click(object sender, EventArgs e)
        {
            OpenFileDialog dialog = new OpenFileDialog();
            if (DialogResult.OK == dialog.ShowDialog(this))
            {
                this.uxJPEGUrlTextBox.Text = dialog.FileName;
            }
        }

        private void uxJPEGProcessButton_Click(object sender, EventArgs e)
        {
            Bitmap jpeg = new Bitmap(this.uxJPEGUrlTextBox.Text);

            foreach (PropertyItem prop in jpeg.PropertyItems)
            {
                this.Print("Tag ID Number= " + prop.Id.ToString());
                this.Print("\r\n");
                this.Print("Type= " + prop.Type.ToString());
                this.Print("\r\n");
                this.Print("Length= " + prop.Len.ToString());
                System.Text.ASCIIEncoding encoding = new
                    System.Text.ASCIIEncoding();
                string manufacturer = encoding.GetString(prop.Value);
                this.Print("Tag Value= " + manufacturer);
                this.Print("\r\n");
                if (prop.Id == 0x5090)
                {
                    this.Print("Contains Luminance table");
                    this.Print("\r\n");
                }
                else if (prop.Id == 0x05091)
                {
                    this.Print("Contains Chrominance table");
                    this.Print("\r\n");
                }

                this.Print("\r\n");
            }
        }

        private void Print(string message)
        {
            this.uxOutputTextBox.Text += message;
        }
    }
}

```

Appendix F: Tags by Group

Tags Inside Groups				
Group	Tag ID	Hexadecimal	Contents	
Uniform Camera	271	0x010F	Uniform Discrete (25% each of 4 values)	
Uniform Camera	65534	0xFFFE	Constant (Uniform)	
Uniform Camera	65535	0xFFFF	Constant (LANL, PS)	
Uniform Camera	20624	0x5090	Luminance Table	
Uniform Camera	20625	0x5091	Chrominance Table	
Uniform Random	65365	0xFF55	Constant (LANL, PS)	
Uniform Random	65520	0xFFF0	Constant (Uniform)	
Uniform Random	65528	0xFFF8	Random Bytes	
Uniform Random	20624	0x5090	Luminance Table	
Uniform Random	20625	0x5091	Chrominance Table	
Twice Tagged	65534 or 65533	0xFFFE or 0xFFFD	Constant	
Twice Tagged	65535	0xFFFF	Constant	
Twice Tagged	20624	0x5090	Luminance Table	
Twice Tagged	20625	0x5091	Chrominance Table	
Taggedjavaimages	65535	0xFFFF	Constant	
Taggedjavaimages	20624	0x5090	Luminance Table	
Taggedjavaimages	20625	0x5091	Chrominance Table	
NormalTest0xff7	271	0x010F	Random Bytes	
NormalTest0xff7	65365	0xFFFF	Constant (LANL, PS)	
NormalTest0xff8	65527	0xFFF7	Box-Muller Integer	
NormalTest0xff9	20624	0x5090	Luminance Table	
NormalTest0xff10	20625	0x5091	Chrominance Table	
Normal0xff9	65365	0xFF55	Constant (LANL, PS)	
Normal0xff10	65521	0xFFF1	Constant (Uniform)	
Normal0xff11	65529	0xFFF9	Discrete Normal (2.3%, 13.6%, 34.1%)	
Normal0xff12	20624	0x5090	Luminance Table	
Normal0xff13	20625	0x5091	Chrominance Table	
Camera Tagged	271	0x010F	Various	
Camera Tagged	65535	0xFFFF	Constant (LANL, PS)	
Camera Tagged	20624	0x5090	Luminance Table	
Camera Tagged	20625	0x5091	Chrominance Table	
CSMLANLPS	271	0x010F	Constant (Doughnut)	
CSMLANLPS	65514	0xFFEA	Uniform Discrete (25% each of 4 values)	
CSMLANLPS	65516	0xFFEC	Random Bytes	

CSMLANLPS	65518	0xFFEE	Weighted (80%, 10%, 7%, 3%)
CSMLANLPS	65523	0xFFF3	Constant (CSMLANLPS)
CSMLANLPS	65526	0xFFF6	Weighted (10%, 15%, 25%, 50%)
CSMLANLPS	65527	0xFFF7	Box-Muller Integer
CSMLANLPS	65529	0xFFF9	Discrete Normal (2.3%, 13.6%, 34.1%)
CSMLANLPS	20624	0x5090	Luminance Table
CSMLANLPS	20625	0x5091	Chrominance Table

Appendix G: Dataset

Dataset			
Group	Size (GB)	Number of JPEGs	Average Size per Image (KB)
Uniform Camera	8.21	38,400	213.72
Uniform Random	8.21	38,400	213.83
Twice Tagged	4.11	19,200	213.8
Taggedjavaimages	4.11	19,200	214.27
NormalTest0xff7	4.1	19,200	213.59
Normal0xff9	16.43	76,800	213.87
Camera Tagged	16.43	76,800	213.88
CSMLANLPS	8.28	38,400	215.6
Blankjavaimages	7.79	38,400	202.81
Total	77.66	364,800	

Appendix H: C# JPEG Generator

//Source: <http://www.codeproject.com/useritems/ImageFileBuilder.asp>

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.Drawing.Drawing2D;
using System.Windows;
using System.Drawing.Imaging;
using System.Drawing;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Set counter, create Bitmap, and save Bitmap
            DateTime beginningSystemTime = DateTime.Now;
```

```

Program pro = new Program();
Bitmap bit = pro.CreateBitmap(100, 100);
bit.Save(@"C:\Users\Parallel Search\testimage.jpg");
DateTime endingSystemTime = DateTime.Now;
Console.WriteLine(endingSystemTime - beginningSystemTime);
Console.ReadLine();
}

public Bitmap CreateBitmap(int width, int height)
{
    try
    {
        Bitmap bmp = new Bitmap(width, height);
        Random rand = new Random();

        // Sets the color of the picture pixel by pixel
        for (int y = 0; y < bmp.Height; y++)
        {
            for (int x = 0; x < bmp.Width; x++)
            {
                // Creates 32 bit depth, where alpha is 255
                Color col = Color.FromArgb(rand.Next(255),
                    rand.Next(255), rand.Next(255));

                bmp.SetPixel(x, y, col);
            }
        }
        return bmp;
    }
    catch (Exception ex)
    {
        ex.GetType();
        return null;
    }
}
}
}

```

Appendix I: Referencing the GD (Google Desktop) API

First and foremost, how did we learn anything of the information presented about GD's API in the body of the report? We could have just read Google's online API descriptions, which we indeed did; but if you compare our descriptions, you will find a few notable discrepancies. The API is given in terms of three IDL files (Interface Definition Language), only one of which we actually used (GoogleDesktopAPI.idl); we might have waded through all that syntax, but we didn't.

IDL files are used to fully define the interfaces of COM objects in a language-independent manner. Their function is to be compiled by the "Microsoft IDL Compiler" (Midl.exe) into a binary interface definition file, called a Type Library (*.TLB files). These can then be directly referenced using #import statements in C++ code, or added as references in Visual Studio VB projects (actually, the IDL files can be added as references in VB projects; Visual Studio VB simply performs the Midl.exe compilation automatically). However, for a C# project, we needed a DLL. Thankfully, Microsoft has tools to import a TLB into DLL augmenting the original interface definitions with meta-data sufficient to use the DLL as a .Net assembly. This tool is TlbImp.exe ("TLB Importer"). (There is also a tool, TlbExp.exe, which exports a .Net assembly into a TLB file.)

So, we compiled all needed IDL API files (which turned out to be only the GoogleDesktopAPI.idl file) into Type Libraries using Midl.exe. Then, we exported the TLB that was compiled into a .Net “interop” DLL - as such exported TLBs are called – using TlbExp.exe.

At this point, just because the interop DLLs exist, doesn’t mean they are usable DLLs in terms of COM. In other words, adding a reference to the GoogleDesktopAPI.dll would allow you to eliminate all compile-time errors. However, when the referencing code that gets instances of any of the contained classes or interfaces runs, there will be a myriad of runtime exceptions. To eliminate these then, we have to register any interop DLLs using RegAsm.exe (“Register Assembly”). Further, with just that, when we run, we will receive an extremely cryptic HRESULT error having something to do with a missing file. The missing file is the DLL because it is not actually in a location where Windows can find it; that place is the Global Assembly Cache, or the GAC. To add it to the GAC, we can use GacUtil. At this point, we can reference the DLL in our .Net project by selecting “Add a Reference” from the Project or other menus in Visual Studio and browsing to DLL you just created. With that, all compile-time and runtime errors with our referencing code will have been resolved.

Step-by-Step

1. Compile the IDL into a TLB with Midl.exe. Command line: ‘midl <COM API>.idl /tlbout:<whatever you want to call it>.tlb’
2. Create a strong name key file. Command line: ‘sn -k <Some arbitrary keyfile name>.snk’
3. Import the TLB into a DLL with TlbImp.exe, signing it with the strong name key file. Command line: ‘tlbimp <whatever you called it>.tlb /keyfile:<That arbitrary name you chose for the keyfile>.snk /out:<whatever you want to call the .Net-consumable DLL>.dll’
4. Register the new assembly with RegAsm. Command line: ‘regasm /register <whatever you called the .Net-consumable DLL>.dll’
5. Add the assembly to the Global Assembly Cache. Command line: ‘gacutil /i <whatever you called the .Net-consumable DLL>.dll’

P.S. You’ll notice that the strong name key file things in step 2 and 3 were never justified; this is because they aren’t understood. The fact is this though, if the project that defines our indexing plugin has a strong name for whatever reason, all DLLs it references will have to have strong names. Therefore, if we don’t give the GD API DLL a strong name, we won’t be able to use it in all projects. Otherwise though, we aren’t at all sure exactly what this does.

Appendix J: Writing a Google Desktop Plugin with .Net

1. Create a class library project.
2. Go to the project’s properties, and do two things:
 - a. Go to the assembly tab and click the Assembly Information button; on the dialog that appears – at the bottom – check the ‘Make assembly COM-visible’ box.
 - b. Go to the Build tab and check the ‘Register for COM Interop’ box. (This simply calls RegAsm on the DLL built for your project, so this step isn’t necessary if you plan to do

this manually. The argument for doing this manually is that you can generate a TLB from your DLL in one step with RegAsm using the /tlb: option.)

3. Open the Add Reference dialog (you can do this several ways; the more consistent way is by selecting 'Add Reference...' from the Project menu), and select the Browse tab. Browse to the DLL that was made from the GoogleDesktopAPI.idl, and add it.
4. For the class you want to be the plugin (i.e. a COM object), assign it a GuidAttribute by typing [Guid("<your GUID>")], where <your GUID> is a valid GUID (which you can obtain using any of many online tools if you don't have a full version of Visual Studio 2005, which we didn't). Whenever you need to register that class as a plugin for anything with GD's API, you'll have to use that string. We recommend copying it into a static string member of your class for easy and reliable referencing.
5. Next, you'll need to have two static methods (privacy doesn't matter for it) that will be called by Windows when the class' DLL is registered and unregistered, respectively. You'll do this by assigning a ComRegisterFunctionAttribute or ComUnregisterFunctionAttribute to a single static method of any class in the project/library/assembly/DLL. This is done by typing '[ComRegisterFunction]' or '[ComUnregisterFunction]' just above the given method. To have access to this attribute, you'll have to add a #using directive for the System.Runtime.InteropServices namespace at the beginning of whichever source file has the class that has the static methods to perform the registration.
6. At this point, it's a matter of implementing the correct registration and unregistration of your plugin in the method specified by the respective COM function attribute. For any kind of plugin, registration starts and ends with calls to the StartRegistration and EndRegistration methods in an instance of the GoogleDesktopRegistrarClass GD API class. In-between however, the process is slightly different. For both, a call to GetRegistrationInterface gets the specific registrar for a query plugin or indexing plugin. When an object of that interface is gotten, then the process changes: For an indexer, RegisterIndexingPlugin is called with the extension of the files the indexer with handle. For a querying plugin, RegisterPlugin is called, returning the integer cookie that then has to be saved.
7. Once this registration has been implemented, the process is entirely different. For an indexer, you simply make sure that the class whose GUID you registered implements DgoogleDesktopFileNotify. And then, you'll have to implement the HandleFile method of that interface, and use the method's 'full_path_to_file' string parameter and 'event_factory' IGoogleDesktopEventFactory parameters to send an event to GD with sufficient descriptive information about the handled file.

Appendix K: Referencing a .Net COM Object in a Native C++ Program

The process of referencing a .Net DLL's public components in native C++ is basically the process of converting Google's API IDLs to DLLs in reverse (performing RegAsm and GacUtil registration), so please refer to Appendix. The notable exception is that we will only go back to a TLB, not all the way to an IDL. That said, once a valid TLB is gotten, we must make sure it is in a directory referenced by the

C++ project. (In Visual Studio C++ Express Edition, this can be done by going to the References section of the project's Property Pages and adding the appropriate path to the 'Reference Search Paths'.)

Step-by-Step

1. Compile your DLL.
2. Register it with RegAsm and create a TLB from it in one step using the following command line:
'regasm <your DLL>.dll /tlb:<your TLB version's name>.tlb'
3. Place the TLB of your assembly in a directory relative to the C++ project you're developing.
4. Add a Path to the Reference Search Paths in the References Property Page of the C++ project.

Appendix L: MPICH2 Software Set Up

These steps assume that Microsoft Visual Studio 2005 Express Edition, the Microsoft Software Development Kit Platform 2, and MPICH2 have been installed on all eight computers.

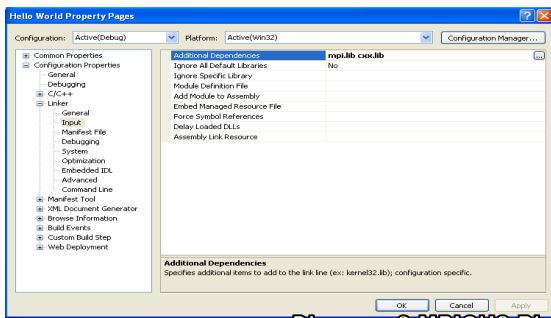


Diagram 3-MPICH2 Bin Add

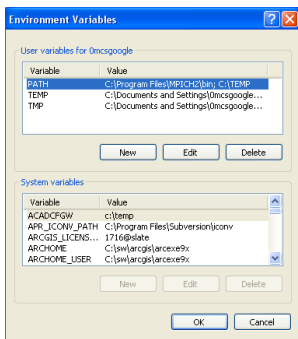


Diagram 4-MPICH2 Linker Add

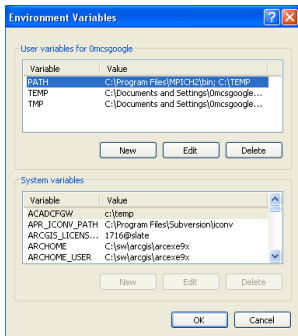


Diagram 5-MPICH2 Set Path

Appendix M: mpiexec Breakdown

The command to run an MPI program:

```
mpiexec -hosts 8 melody 2 minor 2 note 2 rondo 2 meter 2 octave 2 rhythm 2 pitch 2
C:\TEMP\Example.exe
```

- 1) “mpiexec”: Tells the computer that it is running an MPI program. It is the call to run anything that uses MPICH2.
- 2) “-hosts 8”: Tells the computer that there are 8 computers that are going to run the program.
- 3) “melody 2 minor 2 note 2 rondo 2 meter 2 octave 2 rhythm 2 pitch 2”: These are the names of the eight computers. Following each name is the number of

- a. Add the MPICH2 library, bin, and include folders to the Tools->Options->Projects and Solutions->VC++ Directories tab. These files are found in the MPICH2 program file. This step only has to be done once. This can be seen in *Diagram 3*.
- b. The Microsoft Software Development Kit Platform 2 library, bin, and include folders have to be added. These files are found in the Microsoft SDK program file. To set this up, we followed the same steps as in *a*.
- c. Add the linker libraries. This is done by going to Project->Program Properties->Configuration Properties->Linker->Additional Dependencies. In this box type “mpi.lib” and “cxx.lib.” If the above mentioned software has already been installed, then these two library files are already installed on the computer. This can be seen in *Diagram 4*.
- d. Set Up Paths: In order to compile the MPICH2 programs, the mpirun.exe had to be included in the Paths on the computer. To do this, go to My Computer->Properties->Advanced->Environment Variables. Then add the path to the MPICH2 bin and the path to the TEMP folder on the C drive. This can be seen in *Diagram 5*.
- e. The firewall system has to be disabled. This is something that is individual to each computer. For computers running at CSM, this is an administrative task.
- f. The IP address for each computer has to be called dynamically. This was something that we had to have an Administrator do.
- g. Have to include “mpi.h” in the header for every program that uses MPI.

processes or “mini-problems” that that computer is supposed to run. In this case, each computer is running two processes.

- 4) “C:\TEMP\Example.exe”: This is the complete path to the executable file of the program that is going to be run.

Appendix N: MPICH2 Example Code

```
/* -*- Mode: C++; c-basic-offset:4 ; -*- */
/*
 * (C) 2004 by Argonne National Laboratory.
 * See COPYRIGHT in top-level directory.
 */
#include "mpi.h"
#include <iostream>
#include <math.h>
using namespace std;
double f(double);
double f(double a)
{
    return (4.0 / (1.0 + a*a));
}
int main(int argc, char **argv)
{
    int n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI::Init(argc, argv);
    numprocs = MPI::COMM_WORLD.Get_size();
    myid = MPI::COMM_WORLD.Get_rank();
    MPI::Get_processor_name(processor_name, namelen);

    cout << "Process " << myid << " of " << numprocs << " is on " <<
        processor_name << endl;

    n = 10000;                /* default # of rectangles */
    if (myid == 0)
        startwtime = MPI::Wtime();

    MPI::COMM_WORLD.Bcast(&n, 1, MPI_INT, 0);

    h = 1.0 / (double) n;
    sum = 0.0;
    /* A slightly better approach starts from large i and works back */
    for (i = myid + 1; i <= n; i += numprocs)
    {
        x = h * ((double) i - 0.5);
        sum += f(x);
    }
    mypi = h * sum;

    MPI::COMM_WORLD.Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0);
```

```

if (myid == 0) {
    endwtime = MPI::Wtime();
    cout << "pi is approximately " << pi << " Error is " <<
        fabs(pi - PI25DT) << endl;
    cout << "wall clock time = " << endwtime-startwtime << endl;
}

MPI::Finalize();
return 0;
}

```

Appendix O: Mock Set-up Example Code

This is the Driver

```

#include "Object.h"
#include <iostream>
#include <string>
#include <fstream>
#include <cstdlib>
using namespace std;
int main(int argc, char **argv)
{
    ////Create objects////////
    Object zero("bailey", "kelly", 1234567);
    Object one("olen", "davis", 2345678);
    Object two("kari", "macklin", 3456789);
    Object three("shaun", "fischer", 4567891);
    Object four("justin", "cirbo", 5678945);
    Object five("Mr", "Bean", 6789123);
    Object six("Dr", "Jekel", 7894561);
    Object seven("Mr", "Hyde", 9874563);

    //////////MPI Implementation////////

    int rank, size;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int namelen;
    double startwtime = 0.0, endwtime;
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI::Get_processor_name(processor_name, namelen);

    startwtime = MPI::Wtime();

    if(rank==0)
    {
        cout<<"I am process " <<rank<<" of" <<size<<endl;
        cout<<"I am running on " <<processor_name<<endl;
        cout<<"This is my information: " <<endl;
        cout<<zero.get_First()<<" " <<zero.get_Last()<<" " <<zero.get_Phone()<<endl;
    }

    if(rank==1)

```

```

{
    cout<<"I am process "<<rank<<" of"<<size<<endl;
    cout<<"I am running on "<<processor_name<<endl;
    cout<<"This is my information: "<<endl;
    cout<<one.get_First()<<" "<<one.get_Last()<<" "<<one.get_Phone()<<endl;
}

if(rank==2)
{
    cout<<"I am process "<<rank<<" of"<<size<<endl;
    cout<<"I am running on "<<processor_name<<endl;
    cout<<"This is my information: "<<endl;
    cout<<two.get_First()<<" "<<two.get_Last()<<" "<<two.get_Phone()<<endl;
}

if(rank==3)
{
    cout<<"I am process "<<rank<<" of"<<size<<endl;
    cout<<"I am running on "<<processor_name<<endl;
    cout<<"This is my information: "<<endl;
    cout<<three.get_First()<<" "<<three.get_Last()<<" "<<three.get_Phone()<<endl;
}

if(rank==4)
{
    cout<<"I am process "<<rank<<" of"<<size<<endl;
    cout<<"I am running on "<<processor_name<<endl;
    cout<<"This is my information: "<<endl;
    cout<<four.get_First()<<" "<<four.get_Last()<<" "<<four.get_Phone()<<endl;
}

if(rank==5)
{
    cout<<"I am process "<<rank<<" of"<<size<<endl;
    cout<<"I am running on "<<processor_name<<endl;
    cout<<"This is my information: "<<endl;
    cout<<five.get_First()<<" "<<five.get_Last()<<" "<<five.get_Phone()<<endl;
}

if(rank==6)
{
    cout<<"I am process "<<rank<<" of"<<size<<endl;
    cout<<"I am running on "<<processor_name<<endl;
    cout<<"This is my information: "<<endl;
    cout<<six.get_First()<<" "<<six.get_Last()<<" "<<six.get_Phone()<<endl;
}

if(rank==7)
{
    cout<<"I am process "<<rank<<" of"<<size<<endl;
    cout<<"I am running on "<<processor_name<<endl;
    cout<<"This is my information: "<<endl;
    cout<<seven.get_First()<<" "<<seven.get_Last()<<" "<<seven.get_Phone()<<endl;
}

```

```

    if(rank>=8)
    {
        cout<<"There are no more objects"<<endl;
        cout<<"But this process is running on " <<processor_name<<endl;
    }

    if (rank== 0)
    {
        endwtime = MPI::Wtime();
        cout <<"This is how long I've been talking for = " << endwtime-startwtime << endl;
        cout<<"Goodbye"<<endl;
    }

    MPI_Finalize();
    return 0;
    system("Pause");

} // End Main

```

This is the Object.h

```

#pragma once
#include <mpi.h>
#include <iostream>
#include <string>
using namespace std;
class Object
{
public:
    Object(void); //Default constructor
    Object(string first, string last, int phone); //3 Parameter constructor
    string get_First() const {return First;} //First name accessor
    string get_Last() const {return Last;} //Last name accessor
    int get_Phone() const {return Phone;} //Phone number accessor
    void set_First(string first); //First name modifier
    void set_Last(string last); //Last name modifier
    void set_Phone(int phone); //Phone number modifier
    ~Object(void); //Destructor

private:
    string First;
    string Last;
    int Phone;
};

```

This is the Object.cpp

```

#include "Object.h"
#include <string>
Object::Object(void)
{
    First="";
    Last="";
    Phone=3333333;
}
Object::Object(string first, string last, int phone)

```

```

    {
        First=first;
        Last=last;
        Phone=phone;
    }
void Object::set_First(string first)
{
    First=first;
}

void Object::set_Last(string last)
{
    Last=last;
}

void Object::set_Phone(int phone)
{
    Phone=phone;
}

Object::~Object(void)
{
}

```

Appendix P HTTP Request Code

```

#include <cstdlib>
#include <iostream>

using namespace std;

/*
 * Notes:
 * This source demonstrates sending HTTP POST request to webserver from C++
 * This uses sockets hence can be compiled on Linux, UNIX, Win
 */

#define LINUX_OS
#define WIN_OS
#define _DEBUG_PRINT(X) X

//For commn
#include <iostream>
#include <string>
#include <stdlib.h>
#include <assert.h>

#ifndef LINUX_OS
#include <netdb.h>
#endif

#ifndef WIN_OS
#include <Winsock2.h>
#endif

```

```

#define SEND_RQ(MSG)          /*cout<<send_str;*/ send(sock,MSG,strlen(MSG),0);

using namespace std;
//<exe> hostname api parameters

int main()
{
    //Do request here
} //end main
int request (char* hostname, char* api, char* parameters, string& message)
{
    #ifndef WIN_OS
    {
        WSADATA          WsaData;
        WSStartup (0x0101, &WsaData);
    }
    #endif

    sockaddr_in  sin;
    int sock = socket (AF_INET, SOCK_STREAM, 0);
    if (sock == -1) {
        return -100;
    }
    sin.sin_family = AF_INET;
    sin.sin_port = htons( (unsigned short)80);

    struct hostent * host_addr = gethostbyname(hostname);
    if(host_addr==NULL) {
        _DEBUG_PRINT( cout<<"Unable to locate host"<<endl );
        return -103;
    }
    sin.sin_addr.s_addr = *((int*)*host_addr->h_addr_list) ;
    _DEBUG_PRINT( cout<<"Port : "<<sin.sin_port<<" , Address : " << sin.sin_addr.s_addr<<endl);

    if( connect (sock,(const struct sockaddr *)&sin, sizeof(sockaddr_in) ) == -1 ) {
        _DEBUG_PRINT( cout<<"connect failed"<<endl );
        return -101;
    }

    string send_str;

    SEND_RQ("GET ");
    SEND_RQ(api);
    SEND_RQ(" HTTP/1.0\r\n");
    SEND_RQ("Accept: */*\r\n");
    SEND_RQ("User-Agent: Mozilla/4.0\r\n");

    char content_header[100];
    sprintf(content_header,"Content-Length: %d\r\n",strlen(parameters));
    SEND_RQ(content_header);
    SEND_RQ("Accept-Language: en-us\r\n");
    SEND_RQ("Accept-Encoding: gzip, deflate\r\n");
}

```



```

SEND_RQ("Host: ");
SEND_RQ("hostname");
SEND_RQ("\r\n");
SEND_RQ("Content-Type: application/x-www-form-urlencoded\r\n");

//If you need to send a basic authorization
//string Auth = "username:password";
//Figureout a way to encode test into base64 !
//string AuthInfo = base64_encode(reinterpret_cast<const unsigned char*>(Auth.c_str()),Auth.length());
//string sPassReq = "Authorization: Basic " + AuthInfo;
//SEND_RQ(sPassReq.c_str());

SEND_RQ("\r\n");
SEND_RQ("\r\n");
SEND_RQ(parameters);
SEND_RQ("\r\n");

_DEBUG_PRINT(cout<<"####HEADER####"<<endl);
char c1[1];
int l,line_length;
bool loop = true;
bool bHeader = false;

while(loop) {
    l = recv(sock, c1, 1, 0);
    if(l<0) loop = false;
    if(c1[0]=='\n') {
        if(line_length == 0) loop = false;

        line_length = 0;
        if(message.find("200") != string::npos)
            bHeader = true;
    }
    else if(c1[0]!='\r') line_length++;
    _DEBUG_PRINT( cout<<c1[0]);
    message += c1[0];
}

message="";
if(bHeader) {

    _DEBUG_PRINT( cout<<"####BODY####"<<endl);
    char p[1024];
    while((l = recv(sock,p,1023,0)) > 0) {
        _DEBUG_PRINT( cout.write(p,l));
        p[l] = '\0';
        message += p;
    }

    _DEBUG_PRINT( cout << message.c_str());
} else {
    return -102;
}

```

```
#ifdef WIN_OS
    WSACleanup();
#endif
```

```
return 0;
}
```

Appendix Q Unique URL Retriever

```
using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.Text;
```

```
namespace GDRRegistryHacking
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            object searchUrl =
Registry.CurrentUser.OpenSubKey("Software").OpenSubKey("Google").OpenSubKey("Google
Desktop").OpenSubKey("API").GetValue("search_url");
            Console.WriteLine(searchUrl.ToString());
            Console.ReadLine();
        }
    }
}
```

Appendix R: Unique URLs for HTTP Request

Note: http://127.0.0.1:4664/search&s=-XbDScjJXEisMq4lin8li_nm0?q=

Octave: <http://127.0.0.1:4664/search&s=NJbUo9ngkwZpq7eaL.fapIZXeEI?q=>

Pitch: <http://127.0.0.1:4664/search&s=uap4fzzZLwrVoNM1wkBiIczQ4J0?q=>

Rhythm: <http://127.0.0.1:4664/search&s=93MNnKlGmD-9nDYMBZy-Rq5NMQ4?q=>

Rondo: http://127.0.0.1:4664/search&s=pjO6m6g-oSZ1_7y8JkYpgDEOdSI?q=

Minor: <http://127.0.0.1:4664/search&s=fX15IYYHSWlIpcqxNAPApkQgHz0?q=>

Meter: <http://127.0.0.1:4664/search&s=Jg5F6bH0LtfGroJ12hzv4XJZzlU?q=>

Melody: <http://127.0.0.1:4664/search&s=EPaDVCEHMbuaA2nJst98PHtuFkc?q=>

Appendix S: MPICH/Google Desktop Integration code

```
#include "mpi.h"
#include <iostream>
```

```
#import "ParallelSearchComponents.tlb" named_guids
#include <string>
```

```
using namespace std;
```

```
void Query();
```

```

int main(int argc,char **argv)
{

//////////MPI Implementation//////////
int rank, size;
char processor_name[MPI_MAX_PROCESSOR_NAME];
int namelen;
double startwtime = 0.0, endwtime;
MPI_Init( &argc, &argv );
MPI_Comm_size( MPI_COMM_WORLD, &size );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI::Get_processor_name(processor_name,namelen);

startwtime = MPI::Wtime();

if(rank==0)
{
cout<<"I am process "<<rank<<" of"<<size<<endl;
cout<<"I am running on "<<processor_name<<endl;
cout<<"This is my information: "<<endl;

Query();
}

if(rank==1)
{
cout<<"I am process "<<rank<<" of"<<size<<endl;
cout<<"I am running on "<<processor_name<<endl;
cout<<"This is my information: "<<endl;

Query();
}

if(rank==2)
{
cout<<"I am process "<<rank<<" of"<<size<<endl;
cout<<"I am running on "<<processor_name<<endl;
cout<<"This is my information: "<<endl;

Query();
}

if(rank==3)
{
cout<<"I am process "<<rank<<" of"<<size<<endl;
cout<<"I am running on "<<processor_name<<endl;
cout<<"This is my information: "<<endl;

Query();
}

if(rank==4)
{
cout<<"I am process "<<rank<<" of"<<size<<endl;
cout<<"I am running on "<<processor_name<<endl;
cout<<"This is my information: "<<endl;
}
}

```

```

    Query();
}

if(rank==5)
{
    cout<<"I am process "<<rank<<" of"<<size<<endl;
    cout<<"I am running on "<<processor_name<<endl;
    cout<<"This is my information: "<<endl;

    Query();
}

if(rank==6)
{
    cout<<"I am process "<<rank<<" of"<<size<<endl;
    cout<<"I am running on "<<processor_name<<endl;
    cout<<"This is my information: "<<endl;

    Query();
}

if(rank==7)
{
    cout<<"I am process "<<rank<<" of"<<size<<endl;
    cout<<"I am running on "<<processor_name<<endl;
    cout<<"This is my information: "<<endl;

    Query();
}

if(rank>=8)
{
    cout<<"There are no more objects"<<endl;

    Query();
}

if (rank== 0)
{
    endwtime = MPI::Wtime();
    cout <<"This is how long I've been talking for = " << endwtime-startwtime << endl;
    cout<<"Goodbye"<<endl;
}

MPI_Finalize();

system("Pause");
return 0;

system("Pause");

} // End Main

void Query()
{
    HRESULT hRes = S_OK;
    CoInitialize(NULL);
}

```

```

// Declare the reference to the Searcher plugin
ParallelSearchComponents::ISearcher *pManagedInterface = NULL;

// Get an instance of the Searcher plugin (CLSID_Searcher), which implements the ISearcher interface (IID_ISearcher)
hRes = CoCreateInstance(ParallelSearchComponents::CLSID_Searcher, NULL, CLSCTX_INPROC_SERVER,
    ParallelSearchComponents::IID_ISearcher, reinterpret_cast<void**> (&pManagedInterface));

if (S_OK == hRes) // If the instance was got
{
    BSTR query = L"12001";
    // Call public methods of the Searcher class (defined by the ISearcher interface)
    BSTR results = pManagedInterface->StringResultsSearchSimple(query);
    long resultCount = pManagedInterface->ResultCountSearchSimple(query);
    printf("The string results are '%s'\r\n",results); // Doesn't work because of BSTR
    printf("The number of results is %d\r\n", resultCount);
    pManagedInterface->Release();

    system("pause");
}
else
{
    cout << "HRESULT: " << hRes << endl;
    system("pause");
}

CoUninitialize (); //DeInitialize all COM Components
}

```