Team Uxmal – Needlefinder Project Report

Kevin Higgins – *Yuknoom Ch'een* I
David Kluver – *Tuun K'ab' Hix*
Ed (Seth) Marty – *Aj Took'*

Introduction - Ed (Seth) Marty

This project is no longer under active development. This project parses large amounts of preformatted textual data to find interesting phrases. The program is written in C using MPI to parallelize the problem so that the addition of more processors decreases total running time with an arbitrarily large data set.

Specifically, this program will attempt to find the phrases that have the most meaning imbued in them by the language in a given news article or other text. For example, it would find the phrase “If it does” much less interesting than “Aliens invade Earth”.

The data garnered by this process can be used in many ways. For example, the distinction between interesting and uninteresting phrases helps classify documents by their content. The data could be used to specify the topic of an article or document. This is exactly what a search engine needs to do to help a user find a document by subject.

Furthermore, a group of documents may have a phrase in all of them, possibly repeated several times. If that phrase is not a common phrase throughout the entire dataset, the documents would be closely link and would likely be related by the same or a similar subject. Suppose a user found an article on a particular subject as previously explained. Once they have read the article, they could view automatically-related articles to get a broader view of that subject.

Another use would be viewing other subjects that relate to a particular subject. Knowing how a word or phrase relates to other words or phrases helps understand common usage of the word and what it usually applies to.

As a last example: an automatic tag cloud. A manual tag cloud requires tags to be manually assigned to documents, articles, comments, or even blog posts. Based on the number of times a particular tag is used, that tag in the cloud would be larger or smaller, to find the common subjects referred to at a particular website. Clicking on that tag would then bring up the nodes (articles, etc.) tagged with that phrase. This program could automatically assign tags to nodes and help create the tag cloud without the necessity of users self-classifying their text. Examples of this are seen on Flickr and del.icio.us. Some poorer versions have been observed which simply analyze the count of a particular word. This usually ends up with “The” as a very large tag in the cloud.

Methods - David Kluver

The implementation of this project evolved as it progressed from alpha to beta and into its final stage. In each of the stages, one processor, processor zero, was set aside as the manager. Other processors were then delegated as workers which were responsible for the processing of given portions of the overall problem. The main tasks of the manager were to implement an efficient division of labor between the other processors, to collect data sent by the workers, and to analyze the results.

After determining the division of labor, the manager uses MPI commands to send this division out to the workers. The division during the alpha stage was by file decomposition. As a result of time constraints this is still the case as of the end of the beta stage. By the completion of the final stage the goal is to use decomposition by line, so that the number of files given as input doesn't make a difference. The workers then take their portion of the problem and call a count function on it. This function finds white space (spaces and end of lines) and uses this information to tokenize the given portion of the data into phrases of different lengths. Each phrase is added to a self balancing tree. If the phrase is already in the tree, then the tree increases the count of that phrase by 1. Upon completing its portion of the problem, the worker breaks down the tree and sends it, using MPI commands, to the manager. As the manager receives each tree, it inserts the data into its master tree, essentially doing a reduction operation. The tree automatically keeps track of the number of nodes (phrases) it contains. In the final implementation of this project, the manager will not count the number of individual phrases but instead focus on the determination of the of interest (as described above) in each of these phrases.

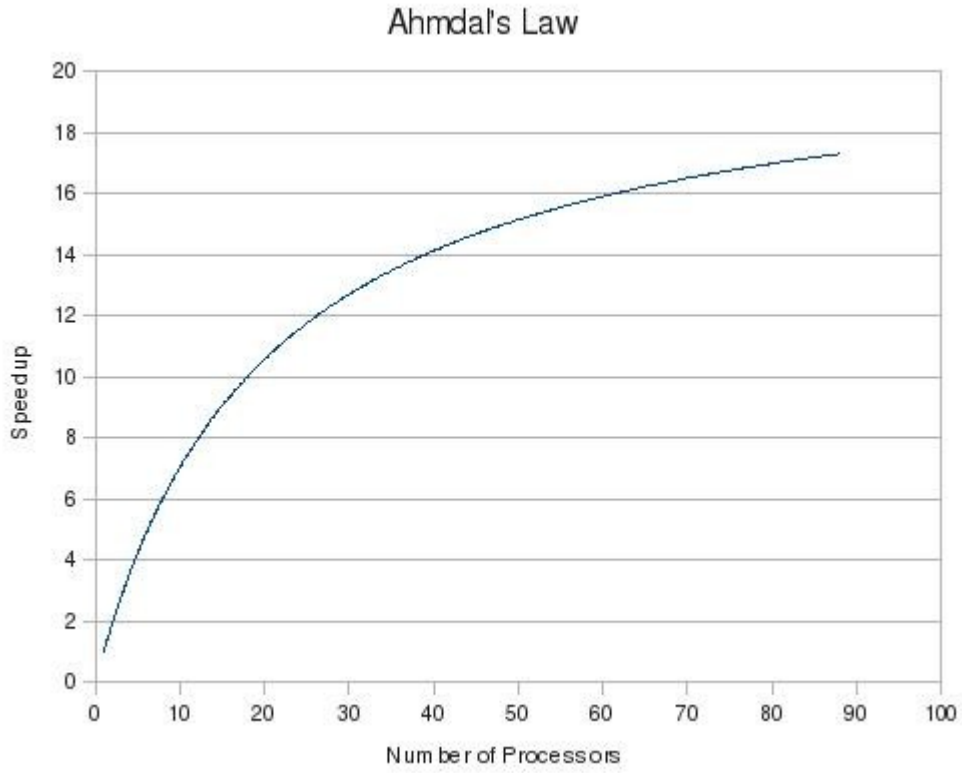
The tree itself is a basic binary search tree that will trigger a tree rotation whenever the tree becomes out of balance. It is a basic AVL tree. When one branch of a subtree becomes longer than the other branch by 2, then the tree is rotated. If the left branch is longer, the tree will rotate to the right, with the left branch's root being the pivot. The essential operations are:
Pivot = Root.OS
Root.OS = Pivot.RS
Pivot.RS = Root
Pivot = Root
Where OS stands for Opposite Side and RS stands for Rotation Side. The last operation is to ensure that the root's parent points to the new root (the pivot).

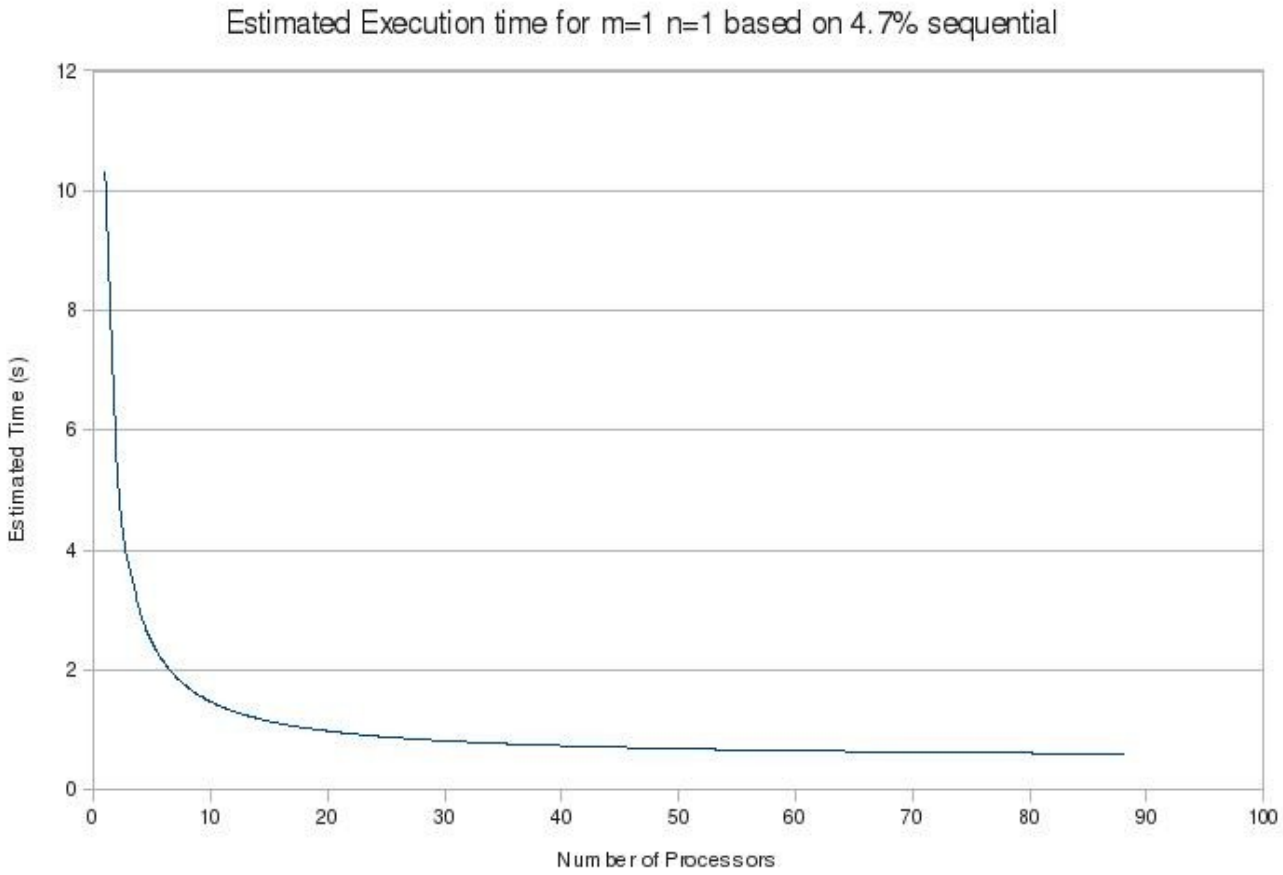
The final stage of the project is not a great leap from the beta stage. The only addition is the calculation of how interesting a phrase is, which is a simple calculation:

I=TF*IDF
TF=Total number of times the node was found
IDF=log2(Total number of articles found in / Total number of articles overall)

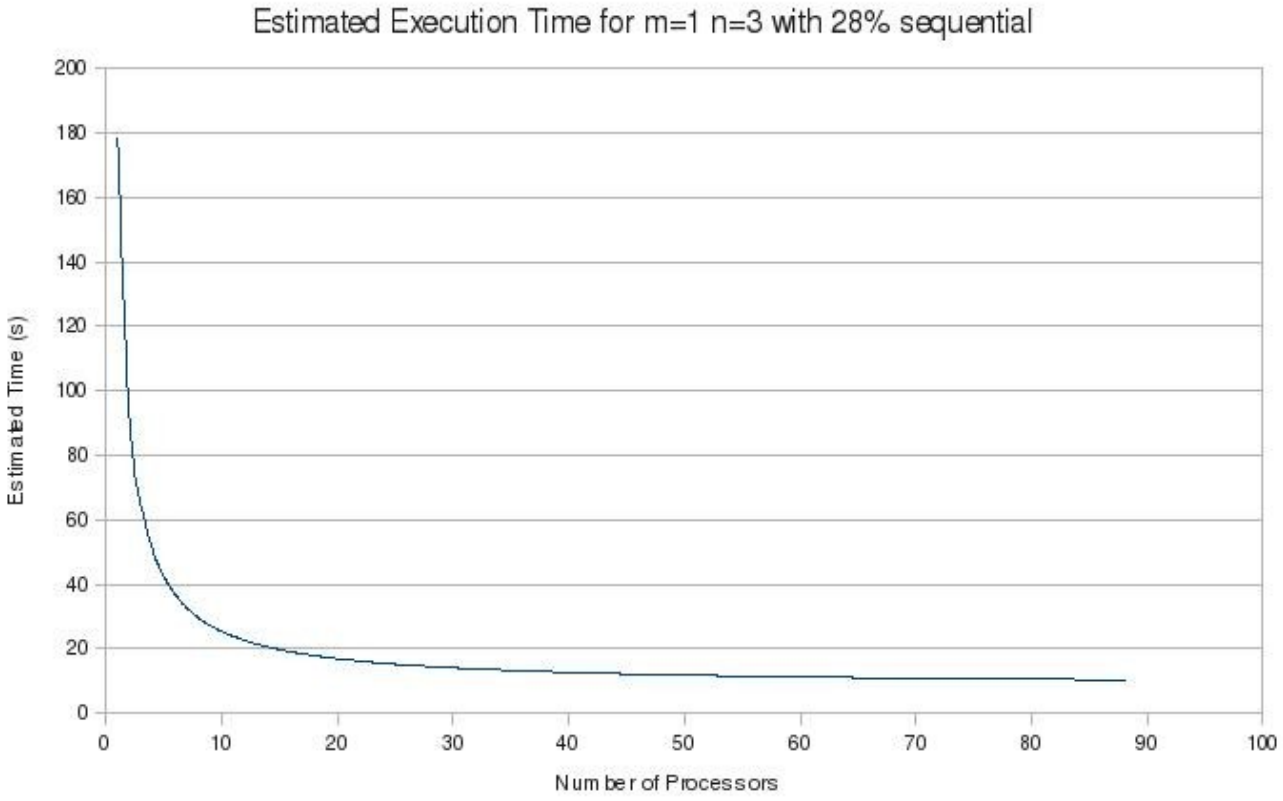
This is distributed among the workers and returned to the manager, which then finds the top r values and displays them (r given on the command line).

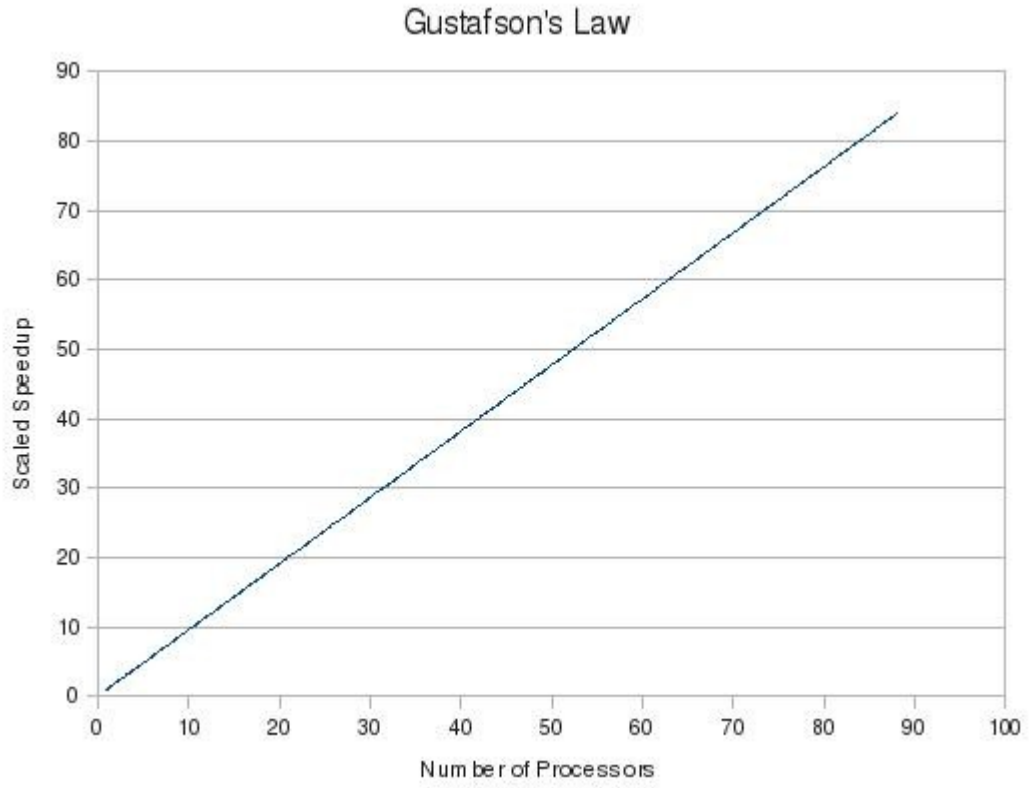
Results – David Kluver




Estimated Execution time for m=1 n=1 based on 4.7% sequential
Estimated Time (s)
Number of Processors
0
2
4
6
8
10
12
0
10
20
30
40
50
60
70
80
90
100

b


Estimated Execution Time for m=1 n=3 with 28% sequential
Estimated Time (s)
Number of Processors
0
20
40
60
80
100
120
140
160
180
200
0
10
20
30
40
50
60
70
80
90
100



These values are based on information obtained running the program on a small dataset. Upon observing the results of these an initial guess was made for the percentage of sequential code. This percentage was estimated to be around 22%. This number was then used in the calculation of Amdahl and Gustafson's laws. These graphs were then analyzed and compared to the data obtained for small scale runs. It was shown that these curves did not accurately portray the speedups seen in the code. Furthermore, it was discovered that the speedups actually differed between the two sets of experimental data. Through the analysis of the experimental data two different values were obtained for the sequential portion of the code, 4.7% and 28%. These percentages were then used with Amdahl's law in order to obtain estimated execution times for varying numbers of processors.

Related works

This section has an overview of some other works that worked on this or a similar problem, and what their approach was.

David Kluver

One common application of problems similar to the previously described is that of translation between languages. In an article written by Li Shao and Hwee Tou Ng, they discus their efforts in such a problem. This project sought to determine English translations for Chinese words given a large quantity

of similar but not identical articles in both languages. The wide availability of works in both languages related to similar topics acted as a guiding force for much of this project. This decision to use similar instead of identical documents not only greatly increased the amount of available data but, in doing so, also greatly increased the size of the problem. By using these documents (“comparable copora”) the team was able to develop methods for determining translations by using not only the individual words themselves but also the context of the words. Although other work had been done with exactly translations of documents and with either individual words or with their context, no project had been completed in which each of these methods was used together. The goals of this project were in many ways similar to that of the project presented in this document. Although this project does not pertain directly to translation between languages, it does deal with a large quantity of documents from a wide variety of sources. Furthermore, the use of the context of the words to aid in the translation is similar to the methods and goals of later stages of this project in which phrases of different lengths will be counted and 'interesting' phrases determined based on a mathematical formula.

In the implementation of this project corpora covering six months (Shao) was used. These articles were then split up into 12 individual sections with each section representing the articles from a half-month period. After partitioning the data in this way, 'interesting' words in the English copora (those appearing no less than 10 times and not appearing in an English-Chinese dictionary of 10,000 words (Shao)) were recorded. These words were then used as potential candidates for translation of Chinese "source words". Although this partitioning scheme is different than that taken in this project, partitioning data chronologically rather than into equal sized sections, the selection of 'interesting' words is similar. This similarity is significant in that although this project was developed to solve a different task, much of its methodology is the same.

Kevin Higgins

This was a very endearing explanation for data mining, "The standard procedure is akin to looking for needles in a needlestack - the problem isn't so much that the desired information is not known, but rather that the desired information coexists with many other valid pieces of information." (Hearst), because it seems to relate so closely to what our project specification is.

A possible method that has been used in the past to analyze text files stems from the interaction paradigm. This is where a program will make a hypothesis on a set of data, and than a user will choose whether or not it is worthy of being checked. This is convenient, because at our current point in time, AI can usually go only so far emulating human preferences.

This would be an interesting vein to look into in regard to interesting phrases, since interesting is such a hard to define term, that it would almost be something required to have a human present to determine. Perhaps a program that is able to 'learn' to limit its scope of a dataset based on past user input would be valuable in such an area. This would be in comparison to simply using how frequent a phrase occurs.

Kevin Higgins

A proposed implementation of an Natural Language Processing (NLP) infrastructure able to handle datasets such as the Gigaword corpus had a fairly detailed specification. The backbone would be made in C++ and use templates for standard data, and configurable classes for dynamic data. In order to then make the best use of the data-structures, python would then be used as an interface in order to rapidly develop tools.

A notable part of the infrastructure lowdown that was found particularly useful was how the article split the NLP infrastructure into components. The list included such things as file processing and text processing which we experienced in the alpha, lexical processing and feature extraction which we will be dealing with soon enough, and than several things that are likely beyond the scope of our project. Such as machine learning methods which is a sort of NLP AI, wherein statistics and decision trees help the program to more ideally interpret text.

It's interesting that the paper says, "Although C++ is extremely efficient, it is not sufficient for rapidly gluing components together to form new tools." (Curran), which seems to entail that C++ is not that bad after all for intensive text searching. Such talk gives a glimmer of hope. Perhaps someday in the future, if we ever again have to write a program to handle large amounts of text, we could use a language aside from C. This would be convenient, because it seems like much of the trouble programming text searching algorithms, is not in the logic, but in doing the coding according to the C creed.

Bibliography - David Kluver

Curran, J. R. 2003. Blueprint for a high performance NLP infrastructure. In *Proceedings of the HLT-NAACL 2003 Workshop on Software Engineering and Architecture of Language Technology Systems - Volume 8* (May 31 - 31, 2003). Human Language Technology Conference. Association for Computational Linguistics, Morristown, NJ, 39-44. DOI= http://dx.doi.org/10.3115/1119226.1119232

Hearst, M. A. 1999. Untangling text data mining. In *Proceedings of the 37th Annual Meeting of the Association For Computational Linguistics on Computational Linguistics* (College Park, Maryland, June 20 - 26, 1999). Annual Meeting of the ACL. Association for Computational Linguistics, Morristown, NJ, 3-10. DOI= http://dx.doi.org/10.3115/1034678.1034679

Shao, L. and Ng, H. T. 2004. Mining new word translations from comparable corpora. In *Proceedings of the 20th international Conference on Computational Linguistics* (Geneva, Switzerland, August 23 - 27, 2004). International Conference On Computational Linguistics. Association for Computational Linguistics, Morristown, NJ, 618. DOI= http://dx.doi.org/10.3115/1220355.1220444