

**EPCC-SS95-04**

**Thesaurus Linguae Graecae (TLG) Parallel Searching**

**Irene A. Moulitsas**

**Abstract**

The search of an ancient language and its literature is a computationally expensive task. Unfortunately the tools that technology has provided are not always efficient. This program aims to make the search of the Ancient Greek Language more efficient, by decreasing the time needed for the search and thus making the task easier for the user.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Document Overview</b>	<b>4</b>
<b>3</b>	<b>Background of Software</b>	<b>5</b>
<b>4</b>	<b>Description of the serial code</b>	<b>5</b>
4.1	Search Types . . . . .	5
4.1.1	Single word . . . . .	5
4.1.2	Two words in a specified range . . . . .	6
4.1.3	Single word using the wildcard character option . . . . .	6
4.2	Modifying the search . . . . .	6
4.2.1	ANY search . . . . .	6
4.2.2	PREFIX search . . . . .	7
4.2.3	SUFFIX search . . . . .	7
4.2.4	EXACT search . . . . .	7
<b>5</b>	<b>Parallelisation</b>	<b>7</b>
5.1	Master task . . . . .	8
5.2	Worker task . . . . .	8
5.3	Sink task . . . . .	8
<b>6</b>	<b>Performance and Results</b>	<b>9</b>
6.1	Single word search results . . . . .	10
6.1.1	First test . . . . .	10
6.1.2	Second test . . . . .	11
6.2	Two word search results . . . . .	12
6.2.1	First test . . . . .	12
6.2.2	Second test . . . . .	13
6.3	Wildcard character search results . . . . .	14
6.3.1	First test . . . . .	14
6.3.2	Second test . . . . .	15
6.4	General Remarks . . . . .	16
<b>7</b>	<b>Conclusion and Future Development</b>	<b>16</b>

## List of Figures

1	Task Farm Implementation . . . . .	9
2	Search for One Word . . . . .	10
3	Search for One Word . . . . .	11
4	Search for Two Words . . . . .	12
5	Search for Two Words . . . . .	13
6	Search using Wildcard Character . . . . .	14
7	Search using Wildcard character . . . . .	15

## List of Tables

1	Timings on the Transputers and the Sun Network.This was timed for 40 texts of 4Kb each with single word search. . . . .	10
2	Timings on the Transputers and the Sun Network.This was timed for various numbers of texts of 4Kb each with single word search on 4 processors. . . . .	11
3	Timings on the Transputers and the Sun Network.This was timed for 40 texts of 4Kb each with two word search. . . . .	12
4	Timings on the Transputers and the Sun Network.This was timed for various numbers of texts of 4Kb each with two word search on 4 processors. . . . .	13
5	Timings on the Transputers and the Sun Network.This was timed for 40 texts of 4Kb each with wildcard character search. . . . .	14
6	Timings on the Transputers and the Sun Network.This was timed for various numbers of text s of 4Kb each with wildcard character search on 4 processors. . . . .	15

# 1 Introduction

The TLG is a collection of electronic texts available on CD-ROM. The texts run to some 60 million words and take up about 550 megabytes. They comprise most of Ancient Greek literature as well as many other texts. Many authors are included starting from ancient years through to the Church Fathers and the Classicists. Also the TLG CD-Rom includes various indexes of the words that exist in the passages. These indexes are helpful in some kinds of search.

The texts are of interest to researchers from various fields. Linguists can find out which other authors use an unusual word. Also they can find out how big the vocabulary range that an author uses is, just by counting the instances of certain words in a predefined sample. Historians use it so as to find all the references of a particular event or topic they might be interested in. Also philosophers can use it to find instances of key phrases in various texts. The texts are also of interest to Classicists, experts on literature and language, and to those studying biblical Greek and the Greek Church Fathers.

At present the texts can be searched either on a specialist micro called Ibycus or on a PC using special software. These searches are quite good for a small group of authors but can take 40 minutes or more to search the whole corpus and that is at best using the Ibycus.

Parallel searching should allow the researcher a much faster time. The main aim of searching is to match a set of words or part-words. It is also sometimes useful to search via the index, especially where rarer words are involved.

## 2 Document Overview

In section 3 is discussed the background of the software. We refer to some similar programs and we make a brief description of them.

In section 4 we have described the serial code that we have written. We refer to the search types that are enabled, as well as the modifications that can be made to a search.

In section 5 there is an overview of the idea of parallelisation of the serial code. The idea of the task farm is roughly described in conjunction with the reasons why we chose task farm.

In section 6 some results of the performance of the parallel code are given, and are compared with the performance of the serial code.

## 3 Background of Software

There are already some similar sequential programs.

The one that was available to us was called 'Musaios'. It runs on PC Window Environment. It allows the user to search the texts available on TLG CD-Rom, and also can search through an index of words. The user can also define the texts he would like to go through or just search the whole corpus, but this seems to be rather inconvenient because of the slow process of each passage.

Also there is a system called Ibycus, running on specialist hardware. Its performance is better than 'Musaios', but nevertheless it is still very slow, especially when someone wishes to search the whole collection of passages. This program was not available to us in the form of its source code.

Initially it was thought that it would better to port the code of Musaios' into Unix. But it didn't turn out to be a good idea. 'Musaios' is written in C++ and is tightly dependent on the Microsoft Class Library. Also a substantial part of 'Musaios' is written in machine code and this reduces its portability. So after a while we decided that I should write my own serial code and try to parallelise that.

## 4 Description of the serial code

### 4.1 Search Types

The serial code is a basic search engine which implements three types of search. These types are :

- search for a single word,
- search for two words occurring together within a certain distance of each other,
- search for a single word but using the wildcard character '\*'

#### 4.1.1 Single word

What this option does is to enable the user to search in the selected texts for a single word. The way this search is implemented is very simple. The engine goes through the text trying to find there a letter which matches with the first letter of the word that the user is looking for. When such letter is found then it keeps its position in an array of pointers and checks if the next letters of this instance match with the next letters of the word. If the letters match then it assumes that the word is found and then tries to check out if other special specifications for the search are met.

### 4.1.2 Two words in a specified range

In this type of search the user is required to specify the range into which the words are to be found. That means that he will look for word1 and word2 among the range of characters he has specified. In this case the search engine initially looks separately for the two words, keeping track of their positions in the text. When the whole text has been searched, the search engine goes through all these instances, keeping only those which differ only less than the specified range.

### 4.1.3 Single word using the wildcard character option

Here the user gives in a word that includes the '\*'. This '\*' can represent any number of characters inside a word. The user is allowed to use only one '\*' in a word. In this search the engine divides the word into two parts. The first one is the part which is before the '\*' and the second part is the one which is after the '\*'. Then the engine looks for these two parts separately and keeps track of all the instances in the text. Next it tries to find out which of the instances form a word that meets the search requirements and if all the requirements are met the the engine keeps the instance.

## 4.2 Modifying the search

The serial program that we wrote enables the user to choose between four choices for each search he wants. These choices are :

- ANY search,
- PREFIX search,
- SUFFIX search,
- EXACT search.

These search types are similar to those provided by the 'Musaios' Software and are useful when searching for words in a language where the beginning and the end can change, depending on the way the word is being used.

### 4.2.1 ANY search

In the ANY search the user wishes to find in the passage any instances of words that might be slightly different from the word that he has given as an input. The program may find instances of words that may not match neither the beginning nor the end of the word he has given. For example if he puts as an input the word port and chooses the ANY option then the results of the search might as well include instances such as important or portable.

### 4.2.2 PREFIX search

In the PREFIX search the user wishes to find in the passage any instances of words that match the input at the beginning but not necessarily at the end. For example if he gives the word `cat` then the word `catalogue` might be given as a result of the search, but the word `scatter` will not be given as it doesn't start with `cat`. This choice might be of great importance for the Greek texts, due to the Greek Grammar. The Greek grammar has different suffixes for the verbs and the nouns. So it would be difficult to search for a verb or a noun without the flexibility of not taking into consideration the various suffixes that it might have in a context.

### 4.2.3 SUFFIX search

In the SUFFIX search the user wishes to find in the passage any instances of words that match the input at the end but not necessarily at the beginning. For example if he gives the word `national` then the word `international` might be given as a result of the search, but the word `nationality` will not be given as it doesn't end in `national`.

### 4.2.4 EXACT search

In the EXACT search the user wishes to find in the passage only the instances of the word he has given as an input. The program will not find instances of words that differ from the input even by one character. For example if he puts as an input the word `treat` and chooses the EXACT option, then possible instances such as `maltreat` or `treatment` will not be included in the results of the search.

## 5 Parallelisation

After the basic serial 'search engine' was created the aim was to parallelise it. The main need of parallelisation is to speed up the search. The central idea is that of the task farm. The Software that has been used is PUL-TF. This is an abstraction of CHIMP-MPI. PUL-TF is a framework that supports the task farm model.

Task farm includes the

- master,
- workers and
- sink

For each one of the master, workers and sink, there have been special routines made which specify what their role in the task farm is. The master gives out the specified

information to the workers. Then the workers perform the task that is assigned to them and they hand over their results to the sink. The sink processes the information given by the workers and prints out the results.

## 5.1 Master task

The program runs with command line. The master has to parse the arguments and check if they are correct or if there is some necessary information missing. If everything is given correctly then the master performs its role. The role that master has, is a function which packs all the necessary information in a buffer and sends this buffer to one worker at a time. Every time a worker comes and asks for work, the master packs the information giving out a different text each time. The information which is packed in the buffer is the name of the file which has the text to be searched, the word(s) that the user is looking for, the type of search (one word, two words, wildcard character), the special modifications of search that the user has specified (any, exact, prefix, suffix) and finally the range for the search (if needed).

## 5.2 Worker task

The workers are basically the search engine. As soon as the workers get the buffer they start unpacking all the information they need. First of all they open the file and get the text from the disk to the memory. Then they start going through the passage specified to them, searching for the word(s). Each worker keeps in an array the location of every instance of the word(s) specified, as well as the location of the beginning of the correspondent sentence. Every time an instance is found, there is a counter which is incremented. There is a constant number which is the upper limit of the number of instances that are allowed to be found in a text. In case this number is reached, then the rest of the text is not searched. When the worker has gone through the whole text then he packs the information into a buffer and sends this buffer to the sink. The information that needs to be sent out is the name of the file which was searched, the number of instances that were found and the locations of the beginnings of the sentences where these instances existed. A worker asks for another text as soon as he has finished the search that was previously assigned to him. A new text is given to him immediately, and he doesn't need to wait for anything.

## 5.3 Sink task

The sink is the output handler. It gets the information buffer from the workers, and then unpacks all the information. It doesn't receive the text itself, but instead it receives the name of the file and must get the text again into memory. This is done so as to avoid transferring a buffer which will be very large. For every instance that is given to it,

the sink starts printing out from the beginning of the correspondent sentence that the instance was found, up to a certain number of characters.

The idea of the task farm proved to be ideal for this project. It's concept and implementation is illustrated in Figure 1

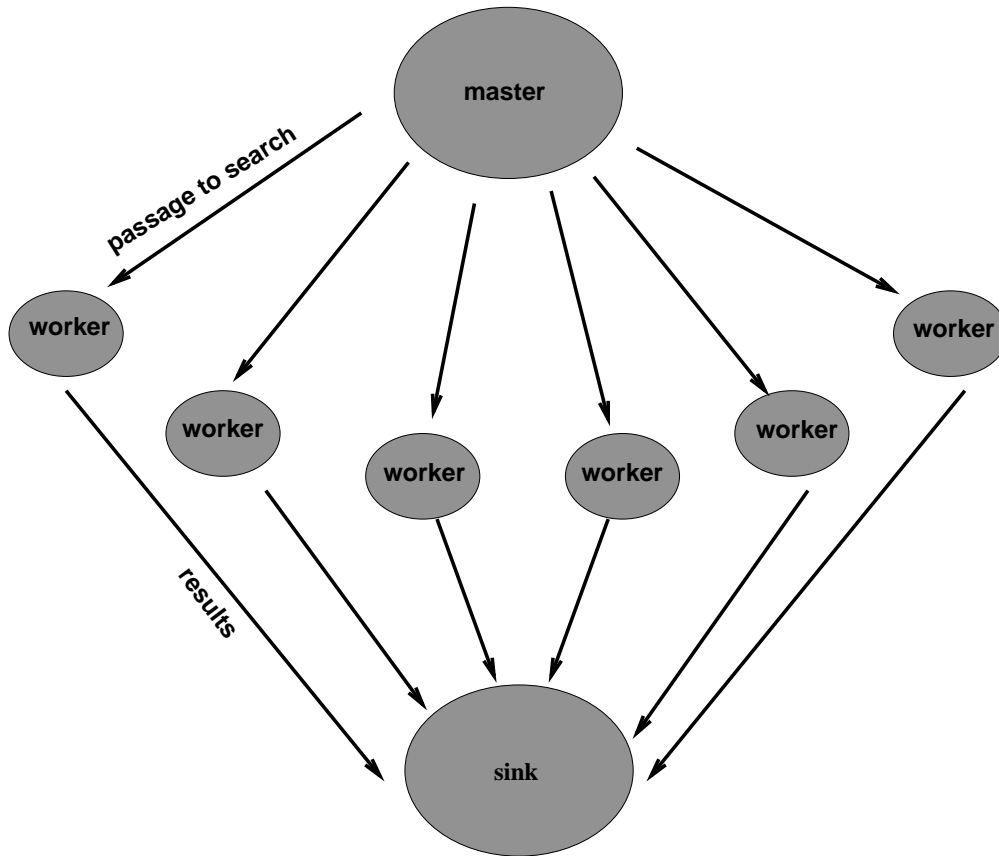


Figure 1: Task Farm Implementation

## 6 Performance and Results

All the timing tests have been made on transputers T800 and a small network made up of 7 Sun4. The following graphs show the performance of the task farm on both architectures. For each type of search there are two performance tables and their correspondent graphs. In each section the first table and graph have been made with a list of 40 texts, each of them being 4 Kilobytes. The results show how execution time is affected by the number of processors. The second table and graph are made on both machines using 4 processors. The results show how execution time is affected by the number of texts that are searched.

## 6.1 Single word search results

### 6.1.1 First test

	Number of Processors					
	1	2	3	4	5	6
Transputers	9.535	6.217	4.538	3.652	4.001	4.120
Suns Hydra	1.942	1.521	1.124	0.907	0.988	1.214

Table 1: Timings on the Transputers and the Sun Network. This was timed for 40 texts of 4Kb each with single word search.

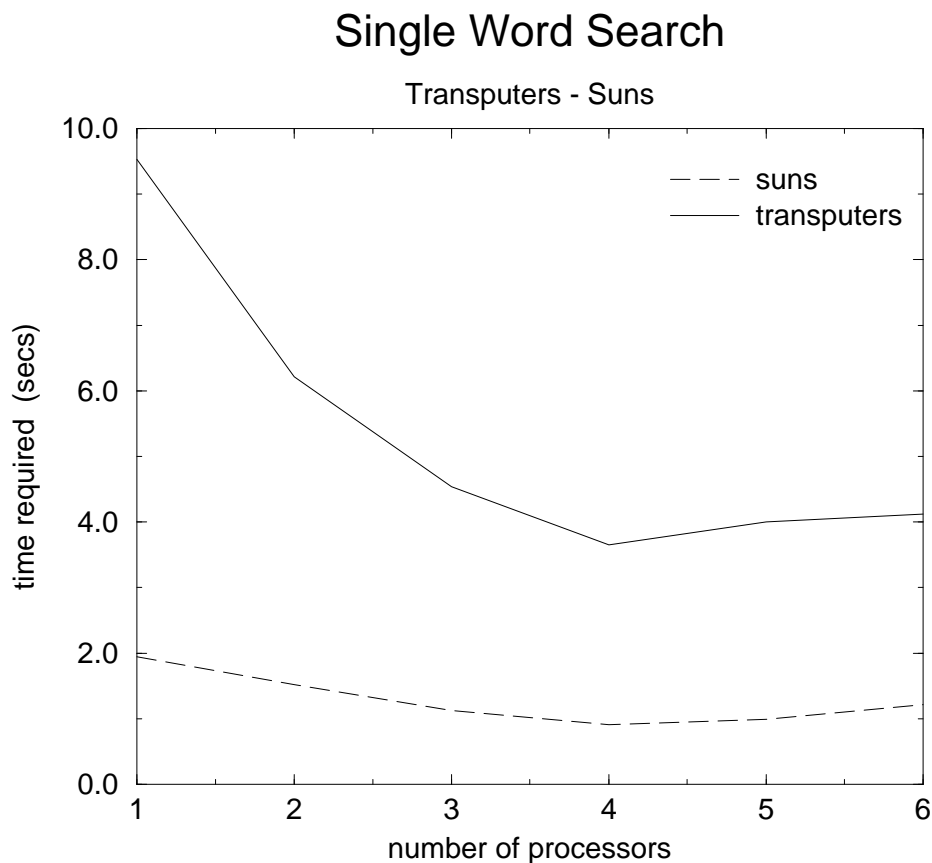


Figure 2: Search for One Word

It is obvious that the Sun Network gives better timings than the transputers. However the speedup at the transputers is much more better. In both cases the performance is better when the number of processors is four. We can also see that in both cases using more than four processors does not make any difference. This can be explained by the fact that we have reached the I/O line. So the time cannot be made any better because after that most of the time is spent on reading the text from the disk to memory.

6.1.2 Second test

	Number of Texts												
	1	5	10	15	20	25	30	35	40	45	50	55	60
Transputers	0.54	1.50	1.65	4.00	3.32	4.44	4.00	4.50	5.12	5.45	5.82	10.54	11.54
Suns Hydra	0.21	0.30	0.33	0.64	0.59	0.67	0.84	0.91	1.37	1.25	1.11	1.23	1.83

Table 2: Timings on the Transputers and the Sun Network. This was timed for various numbers of texts of 4Kb each with single word search on 4 processors.

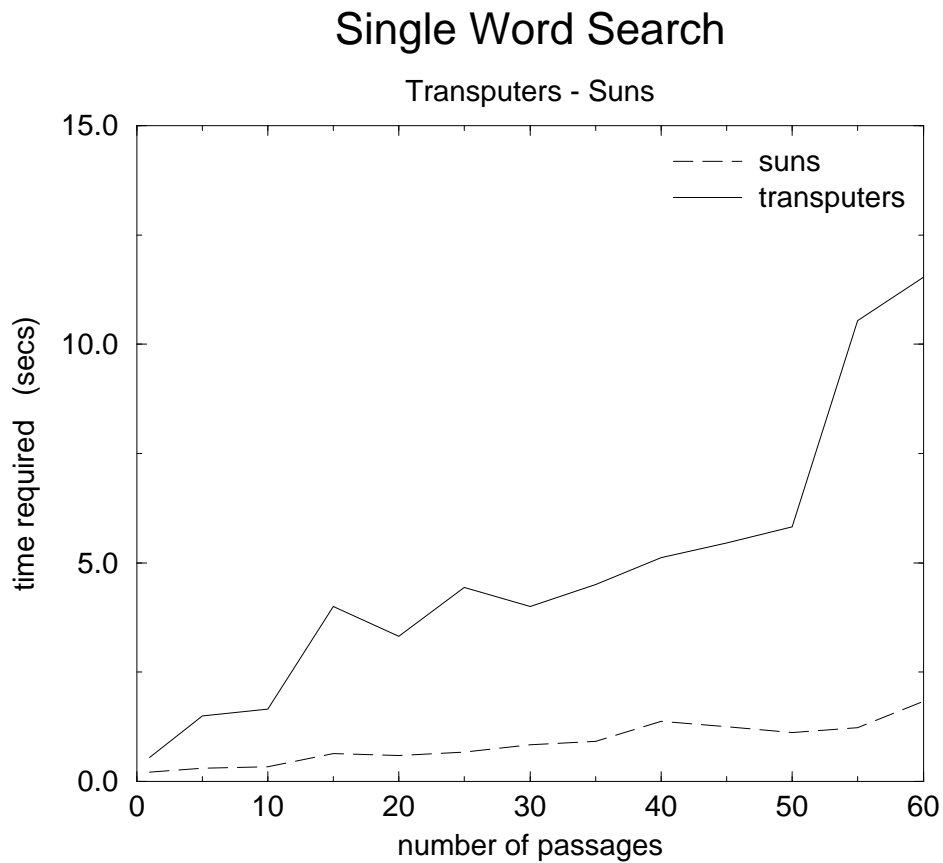


Figure 3: Search for One Word

Performance degrades gracefully on the Sun stations while the performance of transputers exhibits abrupt changes with small variations in the number of passages. This can be explained by the fact that the suns have a wider bandwidth, so they don't need to spend much time on reading the text. On the other hand transputers need more time for reading the text from the disk. So considering that the text must be read both from the workers and the sink, the delay in the performance becomes thus bigger.

## 6.2 Two word search results

### 6.2.1 First test

	Number of Processors					
	1	2	3	4	5	6
Transputers	6.626	6.457	5.993	5.014	4.256	4.372
Suns Hydra	2.165	1.692	1.463	0.869	1.006	1.225

Table 3: Timings on the Transputers and the Sun Network. This was timed for 40 texts of 4Kb each with two word search.

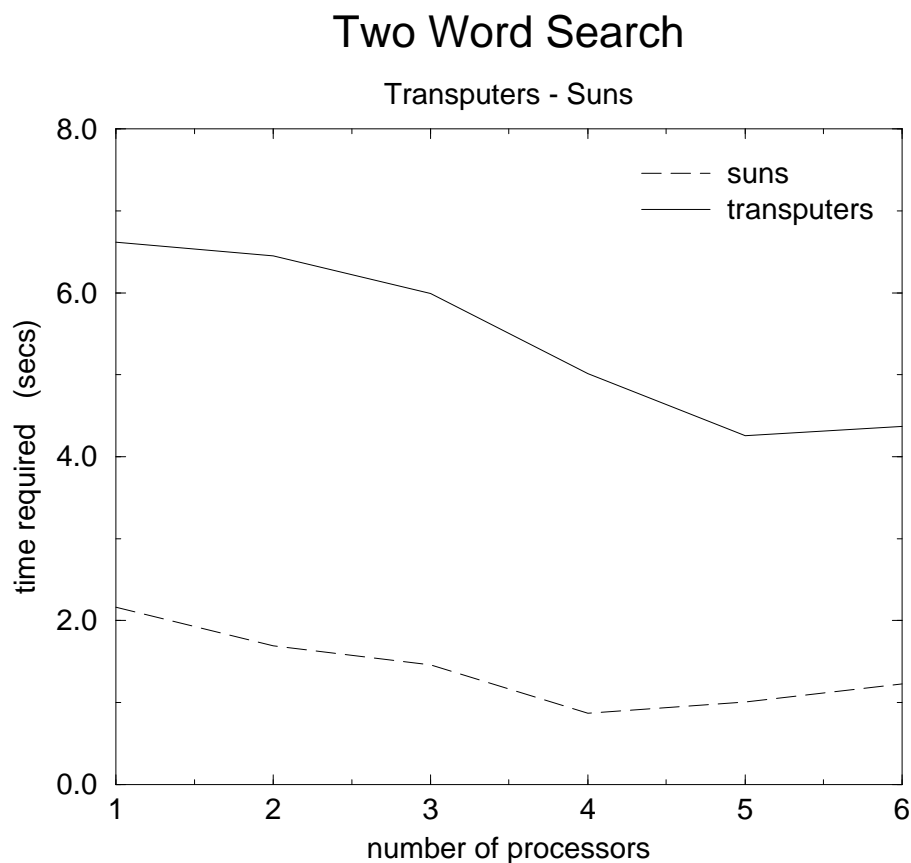


Figure 4: Search for Two Words

This is a more complicated type of search but nevertheless the general idea of the graph is almost the same as in single word search. The Sun Station Network has again a better performance, with best results gained when running on four processors. One would expect that in this search the time should be almost double as much as the time for single word search. But this is not the case of course, as there is not a great difference between

the times of this test and the correspondent test in the previous section. That indicates that the actual time needed for the search is not really much. But instead most of the time is spent on I/O.

**6.2.2 Second test**

	Number of Texts												
	1	5	10	15	20	25	30	35	40	45	50	55	60
Transputers	0.43	1.32	5.50	2.12	3.32	3.54	3.75	5.75	6.63	7.12	10.10	13.00	15.20
Suns Hydra	0.20	0.29	0.36	0.55	0.53	0.76	0.81	0.83	1.31	1.22	1.30	2.21	1.84

Table 4: Timings on the Transputers and the Sun Network. This was timed for various numbers of texts of 4Kb each with two word search on 4 processors.

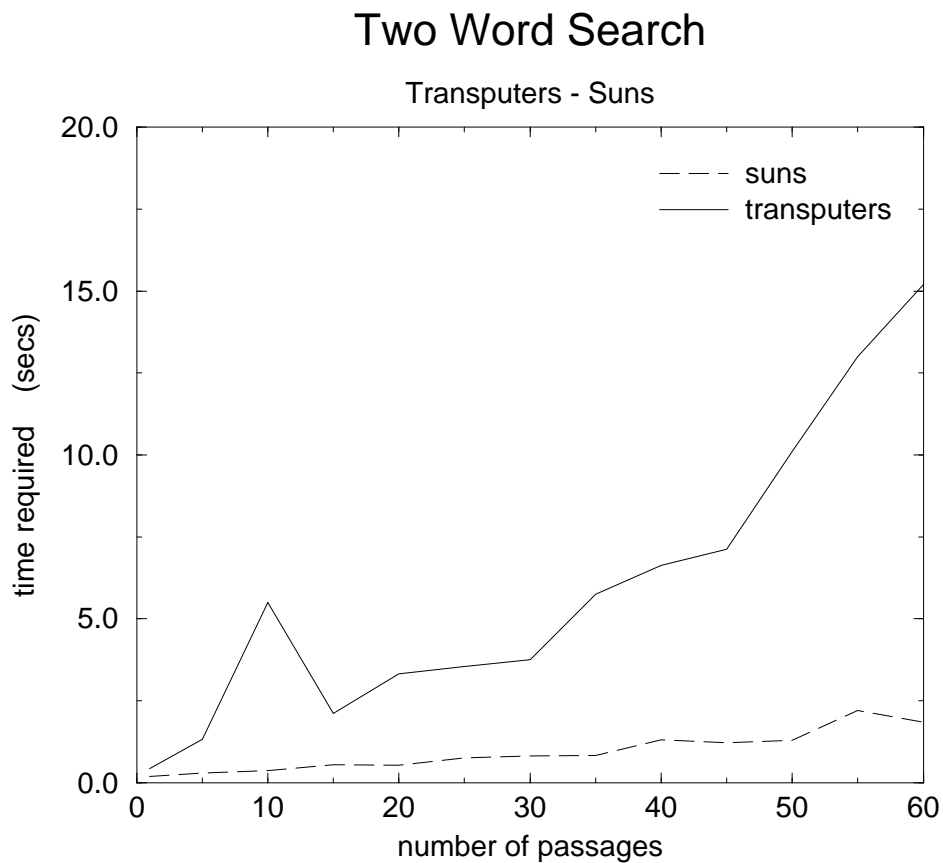


Figure 5: Search for Two Words

Hydra keeps on having much better performance than the transputers. Again Hydra's performance degrades gracefully as opposed to the transputers which do not.

## 6.3 Wildcard character search results

### 6.3.1 First test

	Number of Processors					
	1	2	3	4	5	6
Transputers	8.215	7.325	7.023	4.012	4.123	4.219
Suns Hydra	2.093	1.889	1.580	0.900	1.100	1.200

Table 5: Timings on the Transputers and the Sun Network. This was timed for 40 texts of 4Kb each with wildcard character search.

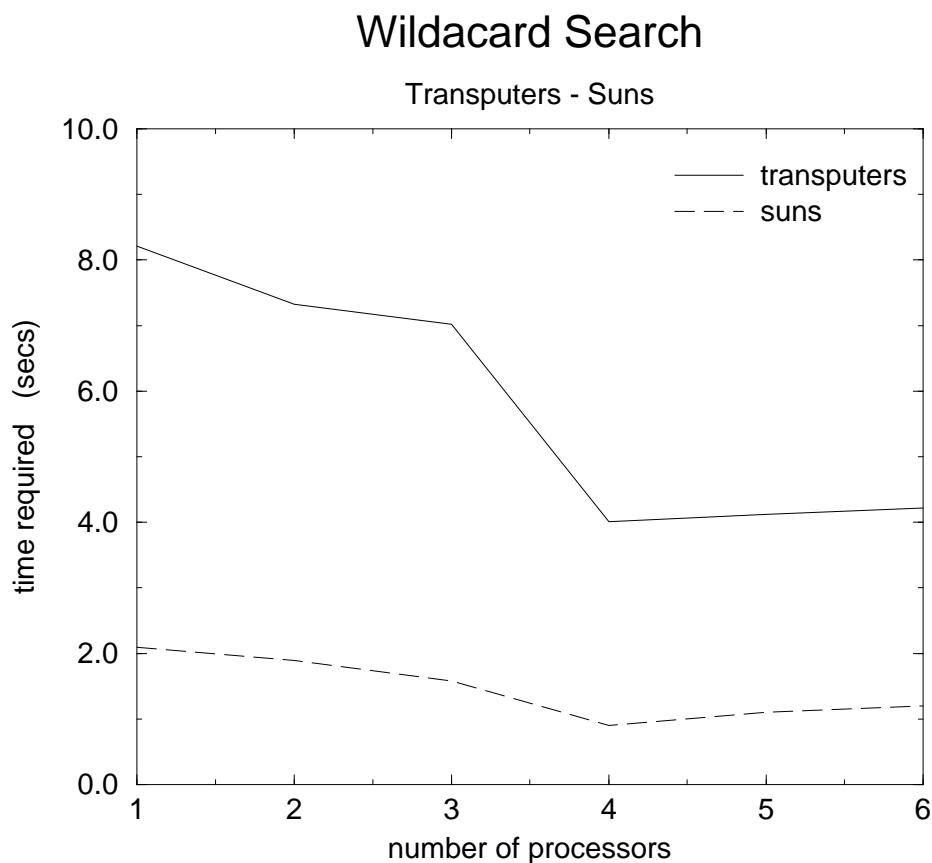


Figure 6: Search using Wildcard Character

There is again the I/O line that the program cannot defeat. The transputers have a nice speedup, acquiring the best results on four transputers. However they are still slower than the Sun network. Hydra does not have a great speedup but its performance is very good. Both systems seem to work perfect on four processors.

6.3.2 Second test

	Number of Texts												
	1	5	10	15	20	25	30	35	40	45	50	55	60
Transputers	0.45	1.66	2.12	2.51	4.12	3.65	5.24	7.02	7.97	7.45	5.78	9.67	7.00
Suns Hydra	0.21	0.28	0.34	0.45	0.98	0.59	0.75	0.84	1.00	0.92	1.01	1.21	1.44

Table 6: Timings on the Transputers and the Sun Network. This was timed for various numbers of texts of 4Kb each with wildcard character search on 4 processors.

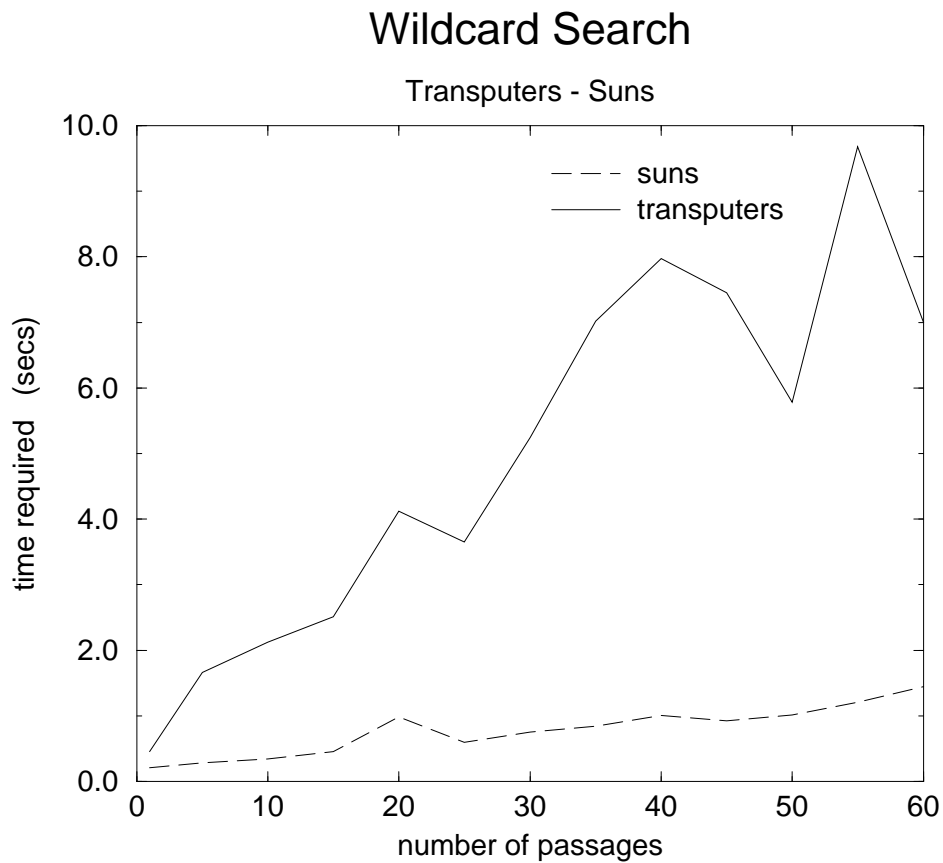


Figure 7: Search using Wildcard character

Performance degrades ungracefully on the transputers. Although the graph is expected to be continuously incremental, there are some unexpected fluctuations observed on the transputers, which cannot be justified. The Sun station network does not exhibit similar problems.

## 6.4 General Remarks

There are some fluctuations in the timings. This is attributed to the fact that 'First Tests' were not done under the same conditions with 'Second Tests.' 'First tests' were done during the night when there were not many users on the Sun network whereas 'Second Tests' were done during the day, when the network was loaded. Also I didn't have the chance to do both tests using the same set of transputers.

## 7 Conclusion and Future Development

The work I did during this summer on this project is just the beginning of a program which can be developed to be more powerful. There are many things which I would like to do if I had more time.

First of all I would like to investigate the possibility of having a bottleneck. I believe that this must be the reason for the fact that the performance is not better when I use more than four processors.

Also I would like to make the search engine more powerful by adding more choices and more capabilities for the search. For example it could search for three or more words in a predefined range. Also it could have 'AND' 'OR' 'NOT' operators. Thus the user could ask to find `word1 AND (NOT word2)`, meaning that he wants to find in the text all instances of `word1` without `word2`.

A future development would be to change the user interface. Now it runs with command line arguments. The arguments that the search engine needs, are:

`[-n]` number of texts to be searched,

`[-t]` type of search (1=one word search, 2=wildcard character use, 3=two word search),

`[-w1]` first word to be searched,

`[-c1]` choice for first word (A=any, E=exact, S=suffix, P=prefix),

`[-w2]` second word to be searched,

`[-c2]` choice for second word (A=any, E=exact, S=suffix, P=prefix),

`[-r]` range where the words can be found.

Note that arguments `[-w2]`, `[-c2]` and `[-r]` are needed only if type of search is specified to be 3.



Irene Moulitsas is an undergraduate student at the Department of Mathematics at the University of Crete in Greece. She will graduate in June 1996 and she would like to pursue graduate studies in Computer Science.

Her areas of interest are Computational Mathematics and in particular Numerical Linear Algebra as well as Computer Science.

This project was supervised by Colin Brough ( EPCC ) and David Mealand ( New Testament Department )