Parallelizing Digital Library Search

G.Narender[#1], Dr. Meda Srinivasa Rao[*2]

Abstract— Digital libraries today are expected to store millions of articles or items of interest. Inexpensive and efficient access to this information becomes critical for success of digital libraries. In order to keep the costs of maintaining digital libraries low, computer programs are taking on the tasks that were traditionally carried out by skilled professional such as librarians. Such tasks often include selection, cataloging and indexing of items in the digital library. Once articles in a digital library are indexed with a set of keywords, computer programs are efficient at retrieving articles that are of interest to a digital library user by resorting to keyword searches of the index database based on user specified keywords. Our research work is investigating ways to speed up implementation and improve experience of using digital libraries. As part of this effort we are investigating ways to use new language extensions such as the Intel Cilk Plus extensions to C and C++ from Intel Corporation that offer a quick, easy and reliable way to improve the performance of programs on multi core processors. Multi core systems are now becoming prevalent on desktops, servers and even laptops. In this paper, we present the results from our work using the Cilk Plus extensions to parallelize digital library searches for articles of interest.

Index Terms— Cilk Plus, Digital Library, Information Retrieval, Keyword Search, Multi Core Systems, System Utilization.

1 Introduction

A digital library is a library in which collections are stored in digital format as opposed to print, microform, or other media. The collections are accessible via computers [1]. The digital content may be stored locally, or accessed remotely via computer networks. We can consider a digital library as a type of information retrieval system. Digital library implementations are very often “automated” to keep the costs of maintaining a digital library under control. The term “automated digital library” is used to describe a digital library whose operation is automated by using computer programs [2]. A question that often comes up is can the experience of using a digital library come close to the experience of using a traditional library. As computers become more and more powerful, the experience of using digital libraries is expected to improve and come close to the experience of using traditional libraries. The improvements in the use of digital libraries are expected to come about from advances in the power of computers driven by Moore's law. Moore's law is the observation that the number of transistors on a semiconductor is expected to double approximately every eighteen months. Stated in other words, computers are expected to become 100 times as powerful in a span of 10 years. This increased power of computers combined with simple or even brute force algorithms can often outperform human intelligence. Computers are also highly efficient at performing monotonous tasks. As an example, they easily outperform humans in brute force searches.

We refer the user to [4, 5] for an overview of the digital library search algorithms. The rest of the paper is organized as follows. Section 2 provides a brief introduction to Intel Cilk Plus most of which has been replicated from our previous work on parallelizing OCR error correction[3]. Section 3 describes our algorithm for the non-parallel digital library search. We basically implement a brute force search mechanism. Section 4 describes our effort to parallelize the brute force search algorithm using Intel Cilk Plus. We give the results of our work in section 5 and conclusions and scope for future work in section 6.

[#1] Research Scholoar, Associate Professor, Department of CSE, Keshav Memorial Institute of technology, Hyderabad ,India.
. *E-mail: guggillanarender@gmail.com*
*[*2] Professor and Director, School of Information Technology JNTUH, Hyde rabad. E-mail:srmeda@gmail.com*

2 Introduction to Cilk Plus

Intel Cilk Plus provides extensions to C and C++ languages that offer an easy and reliable way to improve the performance of programs on multi core processors [8]. Three Intel Cilk Plus keywords provide a simple model for parallel programming. Intel Cilk Plus allows us to write parallel programs using a simple model with only three new keywords to learn. This allows C and C++ developers to move quickly into the parallel programming domain. The three new supported keywords are: _Cilk_for,_Cilk_spawn and _Cilk_sync. The header file <cilk/cilk.h> defines macros that provide names with simpler conventions (cilk_for, cilk_spawn and cilk_sync). A cilk_for loop allows loop iterations to run in parallel and is a replacement for the normal C or C++ for loop. The general cilk_for syntax is:

```
cilk_for (declaration;
                conditional expression;
                increment expression)
```

As an example given the following code, the Cilk runtime system can choose to run different iterations of the loop in parallel.

```
cilk_for (int index = 0; index < 1000; index++)
          func(index);
```

The cilk_spawn keyword is used with a function call and is used to indicate to the Cilk runtime system that the function call can be run in parallel with the caller

As an example, in the following code, the call to function call1() can run in parallel with other_code. The spawned function is usually referred to as the child and the caller is referred to as the parent.

```
void caller() {
        cilk_spawn call1();
        other_code;
}
```

The use of a cilk_sync statement in a function indicates that the current function cannot continue in parallel with its spawned children. The function needs to wait until all the spawned children complete execution before it can continue further.

As an example, in the following code, the parent function caller2 can only start executing the function call to call4 only after the spawned call to call2() completes execution.

```
void caller2() {
        cilk_spawn call2();
        call3();
        cilk_sync;

        call4();
}
```

3 Digital Library Search

As previously mentioned, with ever increasing computer performance, brute force search mechanisms implemented using simple algorithms are expected to improve the experience of using digital libraries. In order to test out our implementation we started out with a word list of about 110000 words. We built up a sample database by randomly associating words from this word list as keywords for articles in the digital library. For our implementation, we assumed a digital library with 5 million articles. Each article was associated with 20 randomly chosen words from the word list as keywords. Our database is essentially a collection of 5 million records where each record contains 20 keywords that were picked randomly from our word list. The search algorithm takes a set of keywords as input from the user and does a search through the database to get articles of interest that are associated with keywords that match the user specified ones. We expect the performance of our search algorithm on this simulated database to reflect real world performance of the same.

The search algorithm reads one record in the database at a time. It then searches through the keywords for this record to make sure that it contains every user specified keyword. If it does, the item index gets added to a list. The search process is repeated for every record in the database. At the end of the algorithm, it reports the indices of articles that match user specified keywords and the amount of time it took to search through the database. The pseudo code for the search algorithm looks as follows:

```
set starting time to current clock counter;
for every record in the database {
        get the keywords that are associated with the article;

        for each user specified keyword {
                if user specified keyword is not in articles keywords {
                        move on to the next record in the database
                }
        }

        add the index of the record to the list of matching indices
}
set ending time to current clock counter;

for each index in the list of matching indices {
        report index of matching article
}

report time taken for search (ending time – starting time)
```

4 Parallel Digital Library Search

For parallelizing the search algorithm we start out by making the observation that searching through the database in the sequential algorithm can be made inherently parallel. We can search through multiple records in the database for user specified keywords in parallel. The search through an article's keywords can be done fully independent of search through another article's keywords. Once we recognize this, parallelizing the search algorithm was simply a matter of using the *cilk_for* loop in place of the *for* loop that iterates through the records in the database. As a result, the pseudo code for the parallel version of our algorithm simply looks as follows:

```
set starting time to current clock counter;
cilk_for every record in the database {
        get the keywords that are associated with the article;

        for each user specified keyword {
                if user specified keyword is not in articles keywords {
                        move on to the next record in the database
                }
        }

        Atomically add the index of the record to the list of matching indices
}
set ending time to current clock counter;
```

```
for each index in the list of matching indices {
          report index of matching article
}

report time taken for search (ending time – starting time)
```

As can be seen from the pseudo code for the parallel version, parallelizing the algorithm was a simple matter of using the cilk_for loop in place of the for loop. In order to avoid any race conditions while adding the index of a matching article into the matching list, we make sure that the list addition operation is done atomically by making use of a lock. We checked the correctness of the parallel version by making sure that indices of the matching articles that get reported are the same in the serial and the parallel version.

5 RESULTS

If In order to measure the speedup from the parallel implementation using Cilk, we measured the time it took to search through the database for the serial and parallel implementations and computed the speedup. Our implementation used the Intel C++ v13.0 compiler. The experiments were done on a system with Intel core i5-2450 processor and 6 GB of memory which can run up to four threads in parallel. We ran the serial and parallel versions of the search algorithms five times and collected the average of the time in seconds it took to complete the search. Our measurements showed an average speedup of 4 when running the parallel version. This demonstrated the ease and effectiveness of using Cilk Plus in parallelizing applications.

6 CONCLUSION AND FUTURE WORK

Because Our work has demonstrated the usefulness of Intel Cilk Plus for parallelizing search algorithms. We were able to demonstrate a significant speedup in the search algorithm by simply resorting to the use of the cilk_for keyword. For future work, we plan to investigate using Cilk Plus extensions to improve other implementation aspects of digital libraries including other retrieval technologies such as semantic searches.

REFERENCES

[1] Greenstein, Daniel I., Thorin, Suzanne Elizabeth. The Digital Library: A Biography. Digital Library Federation (2002)
[2] Arms, William Y, Automated digital libraries: How effectively can computers be used for the skilled tasks of professional librarian ship?, D-Lib Magazine, July / August 2000, Volume 6 Number 7 / 8
[3] G. Narender, Dr. Meda Srinivas Rao. Parallel OCR Error Correction. International Journal of Computer Science and Information Technologies, Vol. 3 (6), 2012,5301-5303
[4] Jiban K Pal, Falguni Pal. Search Algorithms – An Aid to Information Retrieval in Digital Libraries.
[5] Brijesh Shanker Singh. Search Algorithms. DRTC Workshop On Digital Libraries, March 2003, Paper E
[6] http://software.intel.com/en-us/articles/intel-cilk-plus/