Exploiting Coarse Grained Parallelism in Conceptual Data Mining

[Finding a needle in a haystack as a distributed effort]

Mark Blokpoel
Institute for Computing and Information Science,
Radboud University Nijmegen
P.O. Box 9010, 6525 ED
Nijmegen, The Netherlands
blokpoel@acm.org

Franc Grootjen
Nijmegen Institute for Cognition and Information,
Radboud University Nijmegen
P.O. Box 9104, 6500 HE
Nijmegen, The Netherlands
grootjen@acm.org

Egon L. van den Broek
Center for Telematics and Information Technology,
University of Twente
P.O. Box 217, 7500 AE
Enschede, The Netherlands
vandenbroek@acm.org

ABSTRACT

A parallel implementation of Ganter's algorithm to calculate concept lattices for Formal Concept Analysis is presented. A benchmark was executed to experimentally determine the algorithm's performance, including an AMD Athlon64, Intel dual Xeon, and UltraSPARC T1, with respectively 1, 4, and 24 threads in parallel. Two subsets of Cranfield's collection were chosen as document set. In addition, the theoretically maximum performance was determined. Due to scheduling problems, the performance of the UltraSPARC was disappointing. Two alternate schedulers are proposed to tackle this problem. It is shown that, given a good scheduler, the algorithm can massively exploit multi-threading architectures and so, substantially reduce the computational burden of Formal Concept Analysis.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

General Terms

Parallelism, Formal Concept Analysis, Ganter algorithm

1. INTRODUCTION

Searching for information on the world wide web frequently feels like searching a needle in a haystack. To relief the latter feeling, a broad plethora of techniques have been introduced in the field of Information Retrieval (IR). Formal Concept Analysis (FCA) [18] is one of the most appealing techniques since it enables the automatic categorization of items, without loss or approximation errors. Hence, if the needle is in the haystack, it will be found. However, its downside is the enormous amount of possible concepts, and the complexity of the generation process, as expressed through its execution time.

The number of generated concepts used by the FCA may grow exponentially with an increasing number of objects. Fortunately, several studies have shown that in practice the growth is limited to n^3 (e.g., see [12]). Still, for challenging applications, the generation of full blown lattices may take several days.

Despite the incredible expansion of the computer's processing speed in the last decade, efficiency of algorithms is still of the highest interest. A few of the most important reasons for this are 1) the expansion of the information in databases and, especially, of the web, 2) the power consumption (and the heat as its residue) that increased parallel to the processing speeds, and 3) the main memory, which latency is lagging processor performance with at least an order of magnitude. Various techniques have been applied to relief all of the latter problems; e.g., resulting in on-processor caches and out-of-order execution, which optimized at least the processor usage.

To increase the efficiency of nowadays computers, processor manufacturers developed multi-core and multi-threaded architectures. These new architectures exploit memory waiting times and run on lower clock frequencies, reducing power consumption dramatically. However, to effectively make use of the offered parallelism, the software should be suited to run in a parallel fashion. Moreover, this does not reduce the problem of information overload, although it provides the premises to do so.

This article introduces a parallel version of the Ganter algorithm [8], which should restrain the computational burden with FCA through utilizing multi-core and multi-threaded architectures. In Section 2, we briefly describe FCA. Section 3 (re)introduces the original Ganter algorithm followed by the newly developed parallel implementation of the Ganter algorithm in Section 4. A benchmark with three machines, using the parallel Ganter algorithm, is presented in Section 5 followed by its analysis in Section 6. Then, the topic of scheduling is discussed in Section 7. The paper finishes with some final conclusions (Section 8).

2. FORMAL CONCEPT ANALYSIS (FCA)

In 1982, Wille [18] introduced the Formal Concept Analysis (FCA) to enable the conceptualization of data. Obviously this theory has been widely embraced by IR researchers and used to solve quite some IR problems. As shown in [7], FCA can be seen as a Dualistic Ontology, just like Singular Value Decomposition [6, 14].

Various resources on FCA exist on all levels; e.g., [10], [11], [9], and [19]. Therefore, we will now only briefly define FCA.

Let us assume that $\mathcal{D}$ is a set of documents and $\mathcal{A}$ is a set of attributes (words, phrases) that describe these documents. The relation $R \subseteq \mathcal{D} \times \mathcal{A}$ expresses this notion: we write $(d, a) \in R$ iff the document d is described by attribute a. Let $\sim$ be the infix notation for R, so:

$$d \sim a \Leftrightarrow (d, a) \in R$$

We now lift $\sim$ to sets: Let $D \subseteq \mathcal{D}$ and $A \subseteq \mathcal{A}$. Then:

$$d \sim A \Leftrightarrow \underset{a \in A}{\forall} [d \sim a]$$

$$D \sim a \Leftrightarrow \underset{d \in D}{\forall} [d \sim a]$$

Define two polar functions:

$$\mathbf{leftPolar}(D) = \{a \in \mathcal{A} \mid D \sim a\}$$

$$\mathbf{rightPolar}(A) = \{d \in \mathcal{D} \mid A \sim d\}$$

A (formal) concept is a pair (D, A) with:

$$\mathbf{leftPolar}(D) = A \wedge \mathbf{rightPolar}(A) = D$$

The set of formal concepts can be ordered: let (D_1, A_1) and (D_2, A_2) be two concepts. We define

Definition 1

$$(D_1, A_1) \subseteq (D_2, A_2) \iff D_1 \subseteq D_2$$

The set of concepts form a *complete lattice*. The Hasse diagram of this lattice (its transitive reduct) can be used for navigational purposes [15].

Some of the successfull applications of FCA in Information Retrieval are: author identification [17], conceptual document classification, query by navigation and conceptual query expansion [12].

3. GANTER'S ALGORITHM

In 1984, Ganter introduced his algorithm to calculate lattices [8]. In first instance, it was published as a technical report. Surprisingly, it took some time before it became known and its importance was acknowledged; however, since then the algorithm is widely used.

3.1 Sorted power set

The idea behind Ganter's algorithm is to define a total order on all possible document sets. Assume the documents themselves are ordered somehow. For $X, Y \subseteq \mathcal{D}$ we define:

Definition 2 (Lectic order)
$X \leq Y \iff X = Y$ or $max((X \cup Y) \setminus (X \cap Y)) \in Y$

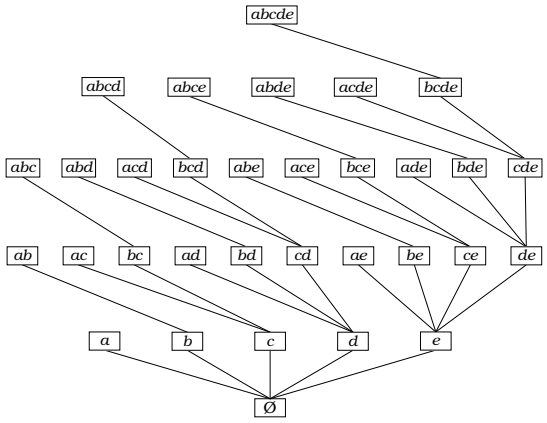

Figure 1: The lectic order for the set of documents $\mathcal{D} = \{a, b, c, d, e\}$.

As seen in [16], we now have the following order on the power set of a given set $\mathcal{D} = \{a, b, c, d, e\}$ with $a < b < c < d < e$

$$\emptyset < a < b < ab < c < ac < bc < abc < \cdots < abcde$$

In order to generate all concepts, Ganter's algorithm will try all document sets in the lectic order. Effectively, it traverses depth-first through a tree (see Figure 1). Notice that the lectic order is compatible with the inclusion order; hence, $X \subset Y$ implies $X < Y$.

3.2 Ganter's ingenuity

The final ingredient for Ganter's algorithm is based on a mathematical proof. This proof states that starting with a node, if we find a concept under certain conditions, it is the smallest one larger than the starting node. Consequently, a lot of subsets can be skipped, which results in a significant speedup.

3.3 The algorithm

A prerequisite for the algorithm is a function that generates the next subset after a current subset D given a $i \in \mathcal{D}$ and $i \notin D$.

Definition 3
Let $D \subset \mathcal{D}$ and $i \in \mathcal{D}$ such that $i \notin D$.

$$\mathbf{nextSet}(D, i) \quad = \quad (D \cup \{i\}) \setminus \{j \in D \mid j < i\}$$

For example, $\mathbf{nextSet}(\{a, b, d, e\}, c) = \{c, d, e\}$.

Theorem 1
Let $D \subset \mathcal{D}$ and $\gamma = \mathbf{RightPolar} \circ \mathbf{LeftPolar}$ the closure operator on the power set of $\mathcal{D}$. If i is the smallest element of $\mathcal{D} \setminus D$ for which $max(\gamma(nextSet(D, i))) \setminus D) \leq i$, then $\gamma(nextSet(D, i))$ is the smallest concept greater then D (with respect to the lectic order).

Using this theorem the Ganter algorithm is able to find all concepts. The algorithm can be expressed in pseudo code, as follows:

```
if(γ(∅) = ∅)
{
  CL = {∅};
}
else
{
  D = ∅;
  I = 𝒟;
  while(D ≠ 𝒟)
  {
    i = smallest element of I;
    I = I \ {i};
    T = nextSet(D, i);
    T' = γ(T);
    if(greatest element of (T' \ D)<=i)
    {
      CL = CL ∩ T';
      D = T';
      I = 𝒟 \ D;
    }
  }
}
```

Legend of symbols:

$\mathcal{D}$	–	set of documents
CL	–	closure set, when the algorithm is finished this set contains all concepts of $\mathcal{D}$
D	–	a subset of $\mathcal{D}$
I	–	$\mathcal{D} \setminus \{i\}$
i	–	a document of $\mathcal{D}$
T	–	next subset of $\mathcal{D}$ containing i: $T = \mathbf{nextSet}(D, i)$
T'	–	Closure of T: $T' = \gamma(T)$

4. DIVIDE AND CONQUER

In the previous section, we have seen that Ganter's algorithm builds an ordered power set that can be considered as a tree. The key issue here is that the concepts found in one branch of the tree do not influence the outcome of the other branches. Furthermore, the function **nextSet** does not require its argument to be a concept. This means that we could split our problem area (the entire power set) into smaller ones that can be calculated concurrently. However, Ganter's original algorithm starts with an empty set and stops when it has reached the complete set. To exploit multi processor machines and calculate small sets concurrently, we aimed to modify it in such a way that it starts and stops with sets of our choice.

4.1 Parallel algorithm

We will present the parallel version of the Ganter algorithm in Java. There are several reasons to implement the algorithm in Java:

- Java has a very lightweight parallel (thread) support with (on some platforms) an extremely efficient hardware implementation [1].

```
JNISet a=startSet.copy();

while(!a.isUniverse())
{
  int i=objectUniverse.getSize()-1;
  for(;;)
  {
    while(a.isElement(i))
      i--;
    a.truncate(i);
    c=a.leftPolar().rightPolar();
    if(!a.smaller(c,i))
      break;
    i--;
  }

  numConcepts++;
  a=c;
  printStream.print(a);
  if(!a.greater(stopSet))
    break;
}
```

Figure 2: The parallel Ganter algorithm in Java.

- The highlevel language constructs of Java are convenient and expressive.
- Thanks to JNI (Java Native Interfacing) low level routines can in implemented in C or machine language.

The variable `c` is used to store the potential concept. `JNISet a` is initialized with the interval's start set. The algorithm will loop until `a` equals the total set (called universe). Different from the original algorithm, the algorithm will also stop when a set greater than the defined stop set is encountered.

1. i is set to the smallest element
2. a is truncated on i
3. check if a is a concept;
   if so: break out of the infinite loop, a concept is found;
   if not: continue with the next element i
4. loop to 1
5. print the concept
6. check if the concept found is greater than the stop set
   if so: break out of the while loop, this sub-problem is solved

Since we are now capable of running Ganter for a sequential subset of $\mathcal{D}$ we should investigate how to make an efficient partition of the powerset of $\mathcal{D}$.

4.2 Scheduling

To run the parallel Ganter algorithm, the total powerset should be split up in sequential intervals (with respect to the lectic order). Unfortunately, the time needed to generate concepts from a certain set is not directly related to the

interval's size. So, to prevent lag from a single thread that consumes too much time, it is smart to create more intervals than number of available threads that can be executed in parallel on the hardware.

The user can specify an arbitrary number of threads to the scheduler. These threads run simultaneously, if possible, managed by the Java Virtual Machine and Operating System. As soon as one thread finishes the scheduler starts a new thread, trying to keep the number of threads as specified at the start. If no more intervals are available the scheduler waits until all threads are finished.

5. BENCHMARK

In contrast with previous adaptions of Ganter's original algorithm that solely provide a formal specification of their work, we chose to run a benchmark to test the algorithm's functioning in practice.

5.1 Benchmark setup

Three hardware architectures were used within the benchmark, namely:

- T1000: UltraSPARC T1 1.0 GHz processor (with 6 cores × 4 strands) with CoolThreads technology; 2GB main memory.
- Athlon64: AMD Athlon64 3000+ (1.8 GHz); 1GB main memory.
- Dual Xeon: Intel Dual Xeon 2.8 Ghz with hyperthreading; 1GB main memory.

As test set for the benchmark we used the Cranfield set [4, 3, 5], which incorporates 1398 technical, scientific abstracts[1]. The collection contains 246174 words, of which 14877 are unique.

To efficiently run tests, we used the following subsets of the Cranfield collection:

- CRAN100: The first 100 documents of the collection, containing 19837 words (2283 unique).
- CRAN200: The first 200 documents of the collection, containing 40723 words (3278 unique).

An important characteristic of the Cranfield collection is that the vocabulary follows Heaps' Law [13]. Figure 3 shows how the vocabulary size varies with the text size. Heaps' Law predicts the vocabulary size to grow sub-linear with the text size. More precise, let x be the text size in words, than the corresponding vocabulary is of size kx^{β}, with k and β text dependent constants. Normally k is between 10 and 100, and β between 0.4 and 0.6 [2]. A very precise fit (i.e., asymptotic standard errors of 0.4366% for k and 0.05893% for β; see Figure 3) yields $k = 7.095$ and $\beta = 0.6167$.

[1]The abstracts are numbered 1 to 1400. Abstracts 471 and 995 are missing.

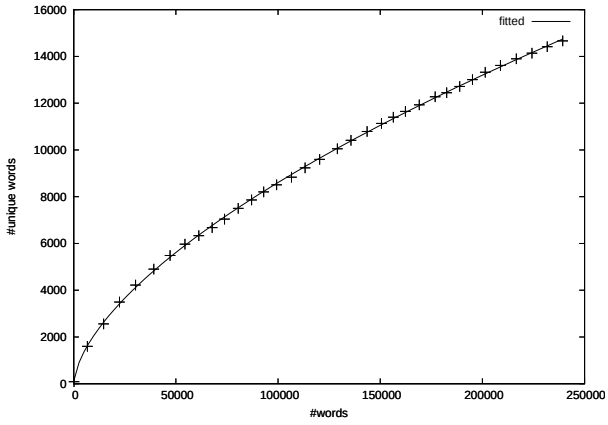

Figure 3: The vocabulary size (#words) of Cranfield's collection in relation to the number of unique words (#unique words). This relation follows Heaps' law.

5.2 Results

The benchmark was executed in two, subsequent, stages. The first run was executed with the CRAN100 set and its results were inspected. Next, the second run was executed with the CRAN200 set and its results were inspected. Both runs were executed with 1, ..., 30 threads. The results are presented in this section and analyzed in the next section.

The results of the first stage are presented in Figure 4, which presents the plots of the results on the Cranfield set of the first 100 documents (CRAN100), yielding 311853 concepts.

As nicely illustrated in the plot of Figure 4, the performance of the Athlon64 machine is determined by its one processor. The number of concepts handled per second is stable for threads 1, ..., 30. The latter is independent of the size of the collection; hence, with the second run its performance would be similar. Henceforth, the Athlon64 machine was excluded from the second run.

The second stage was executed on the Cranfield set of the first 200 documents (CRAN200), yielding 3285489 concepts. The result is presented in Figure 5. Both machines show a similar performance as on the CRAN100 set; cf. Figure 4 and Figure 5.

6. ANALYSIS OF BENCHMARK

To acquire a full understanding of the results found with the benchmark, a further introduction of the machines used (see also Section 5.1) can be useful, as will be provided now. The T1000 machine, as a parallel computer, was designed for throughput, not speed. It uses limited power resources (72W) and runs on 1.0 GHz; however, it should be able to handle 24 threads in parallel using each of the 4 strands on each of its 6 cores. The dual Xeon machine (2.8 GHz) facilitates up to 4 threads with its two processors (power supply: 110W each), each using hyperthreading. Consequently, the Xeon can handle 4 threads in parallel. Moreover, it can do this with a much higher speed than the T1000 machine, due to its relatively fast processors. The Athlon64 machine

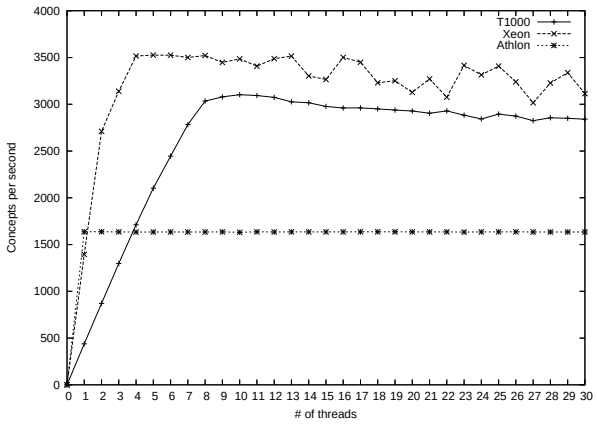

Figure 4: The relation between the number of threads (# of threads) and the number of concepts per second processed as determined through the first stage of the benchmark, with the first 100 documents of the Cranfield collection (CRAN100).

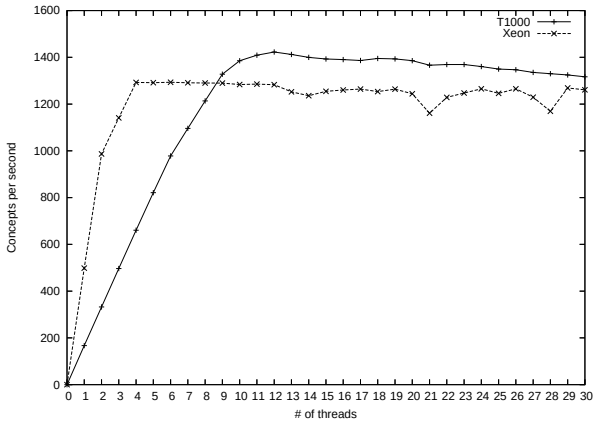

Figure 5: The relation between the number of threads (# of threads) and the number of concepts per second processed as determined through the first stage of the benchmark, with the first 200 documents of the Cranfield collection (CRAN200).

Threads	Athlon concepts/sec.	Xeon concepts/sec.	T1000 concepts/sec.
1	1637	1395	439
2	1637	2710	871
3	1635	3139	1298
4	1634	3516	1711
5	1634	3526	2103
6	1634	3525	2445
12	1635	3488	3074
18	1636	3230	2951
24	1635	3314	2843
30	1635	3114	2840

Table 1: Results of the benchmark on the first 100 documents of the Cranfield collection (CRAN100) with the Xeon and T1000 architectures.

Threads	Xeon concepts/sec.	T1000 concepts/sec.
1	498	167
2	987	333
3	1141	497
4	1293	660
5	1292	821
6	1293	978
12	1283	1423
18	1254	1395
24	1266	1361
30	1261	1317

Table 2: Results of the benchmark on the first 100 documents of the Cranfield collection (CRAN200) with the Xeon and T1000 architectures.

uses one processor with a clock frequency of 1.8GHz (power supply: 89W). AMD claims that the performance of this processor is equal to that of an Intel Pentium processor running on 3.0 GHz, as is denoted with 3000+. However, its efficient processing on this clock frequency does not change that it can handle only 1 thread.

The hardware architecture of the three machines maps nicely on the performance as measured through the benchmark, as described in Section 5.1. In parallel, however, it also raises some questions in particular on the performance of the T1000 machine.

The performance of the T1000 rises almost linear with the number of threads. However, this relation only holds up to around 8 threads for the CRAN100 run and 9 threads for the CRAN200 run. From these moments on, the performance of the T1000 rapidly declines to a limit of respectively 3000 and 1400 concepts per second. Taking into account the architecture of the T1000, it was expected that three distinct phases could be identified in its throughput/performance within the benchmark: 1) up to 6 threads, 2) up to 24 threads, and 3) with 25 threads of more. Within the second phase, three subphases were expected: a) up to 12 threads, b) up to 18 threads, and c) up to 24 threads, denoting respectively 2, 3, and 4 threads per core.

The Xeon machine outperformed the T1000 on the CRAN100 set with on average respectively 3400 and 3000 concepts per second processed, as is also shown in Figure 4. On the CRAN200 set, up to 8 threads the Xeon machine was the fastest; with 9 threads and more the T1000 was slightly better than the Xeon (approx. 1400 opposed to 1250 concepts per second processed), see also Figure 5. Hence, the performance of the T1000 lags behind expectation.

The performance of the Xeon and T1000 can be explained through the thread usage of both machines over time. Figures 6 and 7 show the thread usage for the optimal number of assigned threads with the CRAN200 set. Figure 6 shows the decline of T1000' processor usage over time. From 35 seconds on, the number of used threads handled by the T1000 decreases significantly. In contrast, the processor usage of the Xeon machine is constant over time, as is illustrated in Figure 7.

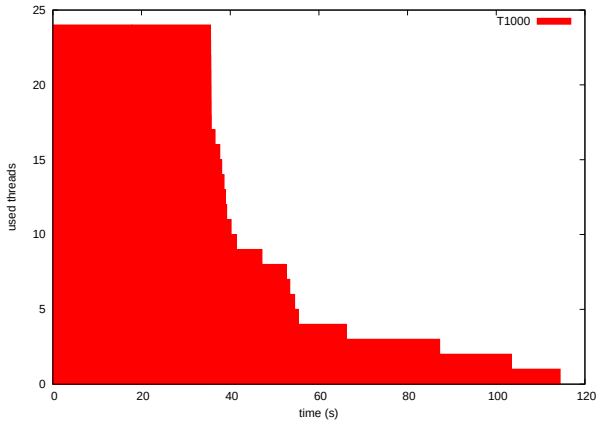

Figure 6: Thread usage of the T1000 (UltraSPARC T1, 1.0 GHz) machine with the first 200 documents of the Cranfield collection executed with 24 threads.

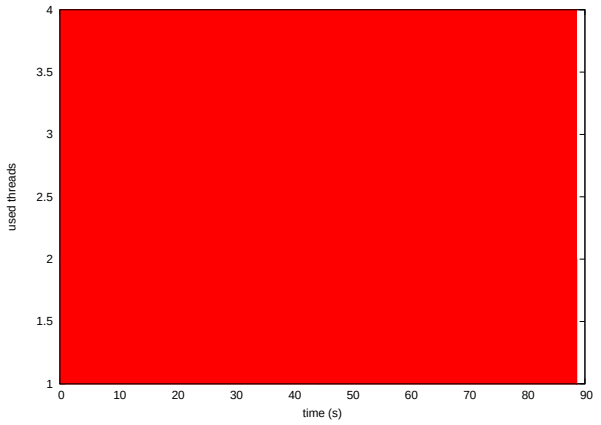

Figure 7: Thread usage of the Intel dual Xeon (2.8 GHz) machine with the first 200 documents of the Cranfield collection executed with 4 threads.

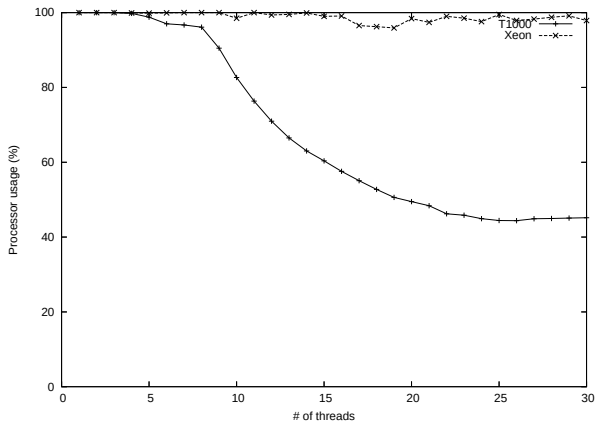

Figure 8: The processor usage of both the T1000 (UltraSPARC T1, 1.0 GHz) and the Intel dual Xeon (2.8 GHz) machine in relation to the number of threads the algorithm was executed with.

Figure 8 shows the processor usage for both architectures, using the current scheduler. As is visualized in Figure 8, the T1000 cannot use all threads during the execution of the program. Consequently, its processor is not optimally used, as is shown in Figure 8. From 8 threads on, the overall average processor usage drops significantly to 44%. The scheduler is not able to assign more than 8 threads efficiently. In contrast, the Xeon, with a maximum of four threads and a higher clock frequency uses all 4 threads almost optimally (99%); hence, the scheduler utilizes all threads (99%).

7. SCHEDULING

Scheduling is of key importance for parallel processing, this is no different for our parallel implementation of the Ganter algorithm, as has been denoted in Section 6. We will now denote what the performance of our algorithm would have been, without the limitation of the scheduler. Subsequently, the steps to be made to develop an efficient scheduler for our algorithm will be described.

7.1 Theoretic optimal performance

To achieve an optimal performance of the parallel Ganter algorithm, 100% of the processing capacity should be used, preferably for all possible number of threads. In Figure 8, the algorithm's usage of the processor during the benchmark on the CRAN200 set is presented. Figure 9 presents the algorithm's performance on the same set in case of perfect scheduling; i.e., a constant 100% processor usage. The application of such an ideal scheduler would improve the performance of the Xenon machine slightly and would reduce the variability of its performance; cf. Figures 5 and Figure 9. In contrast, the scheduler would boost the performance of the T1000 machine; cf. Figures 5 and Figure 9.

At the start of the previous section, the architecture of each of the three machines was discussed. As with the experimental results, the theoretical performance of the algorithm can be explained through the machines' architectures. As denoted before, the performance of the T1000 can be characterized by three phases, with a second phase in which

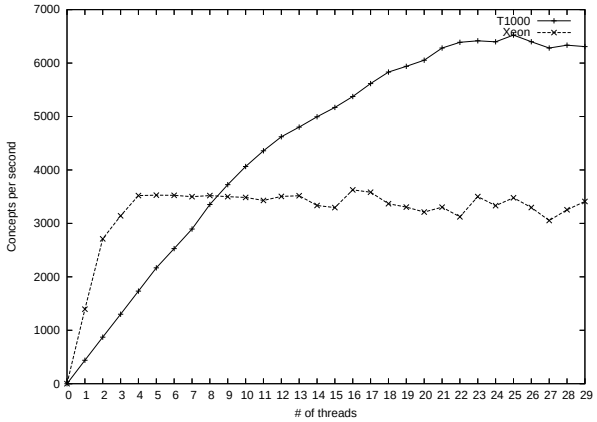

Figure 9: The relation between the number of threads (# of threads) and the number of concepts per second processed in the case of a a constant 100% processor usage, with the first 200 documents of the Cranfield collection (CRAN200).

three subphases can be identified. Altogether, it can be expected that the relation between the number of concepts determined per second and the number of threads allocated changes per step of six threads. The latter is nicely illustrated in Figure 9, which presents a very slowly declining graph. However, the graph mostly illustrates the excellent handling of the 24 threads (through 6 cores) by the T1000 processor. In contrast with the T1000, the Xenon machine is, as hypothesized before, hardly limited by the current scheduler, its performance in the benchmark is near the theoretical optimum.

7.2 Alternative schedulers

As is illustrated by the algorithm's hypothetical performance, using an optimal scheduler, the latter component is essential for the parallel implementation of Ganter's algorithm as introduced in this paper. We will now briefly denote two alternative schedulers that could boost the algorithm's performance in practice: a predictive and a dynamic scheduler.

A predictive version of the current scheduler tries to estimate (beforehand) how long a specific interval will run. If this estimation is longer then a preset value, it will split up the interval accordingly. It is still uncertain whether such an estimation function exists.

A dynamic scheduler just starts threads for reasonable intervals. As soon as there are no more intervals left, and there are still threads running, the scheduler will intervene: it will stop a running thread, notes the progress it made, and will split up the remaining interval and reschedule them to new threads. Obviously this involves some interthread communication, but will lead to optimal load distribution.

8. CONCLUSIONS

A parallel implementation of Ganter's algorithm to calculate concept lattices for Formal Concept Analysis is presented. A benchmark was executed to experimentally determine the algorithm's performance. Three machines were used with 1 (AMD Athlon64), 4 (Intel dual Xeon), and 24 (UltraSPARC T1) threads in parallel and two subsets of Cranfield's collection were chosen as document set. In addition, the theoretically maximum performance was determined. The performance of the UltraSPARC in the benchmark was somewhat disappointing. Scheduling appeared to be the problem. To relief the latter problem, either a predictive scheduler or a dynamic scheduler are proposed. However, the latter solutions are not yet fully functional. Nevertheless, with the latter solution in mind, a promising, highly efficient algorithm is introduced, optimized for parallel machines. Consequently, the algorithm can massively exploit multi-core and multi-threading architectures and so, substantially reduce the computational burden of Formal Concept Analysis.

Acknowledgment

This research was supported by the Netherlands Organization for Scientific Research (NWO) under project number 634.000.018.

9. REFERENCES

[1] SUN Fire T1000 and T2000 Server Architecture: Unleashing the UltraSPARC T1 Processor with CoolThreads Technology. Technical report, Sun Microsystems, Inc., 2005. `http://www.sun.com/servers/coolthreads/coolthreads_architecture_wp.pdf`.

[2] R. Baeza-Yates and G. Navarro. Block-addressing indices for approximate text retrieval. *Journal of the American Society for Information Science*, 51(1):69–82, 2000.

[3] C. Cleverdon. The cranfield tests on index language devices. *Aslib Proceedings*, 19(6):173–194, 1967.

[4] C. W. Cleverdon. ASLIB cranfield research project: report on the first stage of an investigation into the comparative efficiency of indexing systems. Technical report, The College of Aeronautics, Cranfield, England, September 1960.

[5] C. W. Cleverdon. The significance of the Cranfield tests on index languages. In E. Fox, editor, *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12, Chicago, Illinois, United States, October 1991. New York, NY, USA: ACM.

[6] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[7] F. A. Grootjen and Th. P. van der Weide. Dualistic ontologies. *International Journal of Intelligent Information Technologies*, 1(3):1–20, 2005.

[8] B. Ganter. Two basic algorithms in concept analysis. Technical Report FB4-Preprint No. 831, TH Darmstadt, 1984.

[9] B. Ganter, G. Stumme, and R. Wille. Formal concept analysis: Theory and applications. *Journal of Universal Computer Science*, 10(8):926–926, 2004.

[10] B. Ganter and R. Wille. *Formale Begriffsanalyse, Mathematische Grundlagen*. Springer-Verlag Berline, 1996.

[11] B. Ganter and R. Wille. *Formal Concept Analysis:*

Mathematical Foundations. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. Translator-C. Franzke.

[12] F. A. Grootjen. *A Pragmatic Approach to the Conceptualization of Language*. PhD thesis, Radboud University, Nijmegen, The Netherlands, 2004.

[13] J. Heaps. *Information Retrieval - Computational and Theoretical Aspects*. Academic Press, Inc., New York, 1978.

[14] E. Hoenkamp. Unitary operators on the document space. *Journal of the American Society for Information Science and Technology*, 54(4):319–325, 2003.

[15] W. Roelofs and F. Grootjen. Navcon, Navigating the conceptual space. In M. M. Dastani and E. de Jong, editors, *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007)*, pages 447–448, Utrecht, the Netherlands, November 2007.

[16] D. Taouil, N. Pasquier, Y. Bastide, and L. Lakhal. Applications and performances computing closed set lattices: Algorithms, applications and performances, 2003.

[17] L. van der Knaap and F. Grootjen. Author identification in chatlogs using formal concept analysis. In M. M. Dastani and E. de Jong, editors, *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007)*, pages 181–188, Utrecht, the Netherlands, November 2007.

[18] R. Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In I. Rival, editor, *Ordered sets*, pages 445–470. D. Reidel Publishing Company, Dordrecht–Boston, 1982.

[19] K. E. Wolff. *A first course in formal concept analysis. How to understand line diagrams*, volume 4, pages 429–438. Stuttgart, Germany: Gustav Fischer Verlag, 1994.