A Parallel Shared-Memory Architecture for OWL Ontology Classification

Zixi Quan and Volker Haarslev
Department of Computer Science and Software Engineering,
Concordia University,
Montréal, Canada
Email: z_qua@encs.concordia.ca, haarslev@encs.concordia.ca

***Abstract*—The *Web Ontology Language* (OWL) plays an important role in the *semantic web* to represent domain knowledge using classes, properties, and individuals. OWL reasoners analyze ontologies and offer inference services such as class satisfiability and subsumption. Ontology classification is an important and widely used service that computes a taxonomy of all classes occurring in an ontology. It can require significant amounts of runtime but most OWL reasoners do not support any kind of parallel processing. We present a novel thread-level parallel architecture for ontology classification that is ideally suited for shared-memory SMP servers, where each thread can be mapped to a core on a one-to-one basis. We evaluated our prototype implementation with a set of real-world ontologies. Our experiments demonstrate a very good scalability resulting in a speedup that is linear to the number of available cores.**

***Index Terms*—OWL; ontologies; classification; parallelism;**

I. Introduction

The Web Ontology Language (OWL) as part of the semantic web [1] is a widely used knowledge representation language for describing knowledge in application domains. Description logics (DLs) [2] are a family of logic-based knowledge representation formalisms, which describe a domain in terms of concepts (classes), roles (properties), and individuals. OWL can be considered as a syntactic variant of a very expressive DL. Typically supported DL inference services are concept satisfiability, concept subsumption, instance checking, query answering etc [3]. The first two services deal only with concept expressions and form the basis of ontology classification.

Tableau-based methods (see Section II) are widely used in ontology reasoners for implementing the above-mentioned inference services. In order to speed up reasoning and improve the effectiveness of reasoners, it is necessary to develop efficient and optimized reasoning techniques to implement inference services. OWL ontology reasoning is known to be N2EXPTIME-complete (NEXPTIME-complete if the property hierarchy can be translated into a polynomially-sized nondeterministic finite automaton) [4]. Although most OWL reasoners are highly optimized quite a few real-world ontologies exist that cannot be classified within a reasonable amount of time.

High performance computing (HPC) methods can offer a scalable solution to speed up OWL reasoning. Our HPC approach is based on parallel reasoning techniques for OWL classification. Compared with sequential OWL reasoners, such as Racer [5], FaCT++ [6], and HermiT [7], parallel OWL reasoners work concurrently and distribute the whole task into smaller subparts to speed up the process. A few OWL reasoners integrated parallelization techniques; Konclude [8] is highly performant but its TBox classification is sequential; ELK [9] supports parallel TBox classification but is restricted to the very small $\mathcal{EL}$ fragment of OWL. Moreover, some other parallel DL reasoning methods have shown promising results in the past few years such as the first parallel approach for TBox classification [10] using a shared-tree data structure, merge classification [11]–[13] implementing parallel divide-and-conquer approaches, and [14] proposing a parallel framework for handling non-determinism caused by qualified cardinality.

Our work is motivated by previous parallel approaches and also expand ideas presented in [15] to parallel processing. Our HPC approach is implemented with a shared-memory architecture, atomic global data structures, and new strategies for parallel subsumption testing. In order to keep our architecture universal we use OWL reasoners as plug-ins for deciding satisfiability and subsumption. Currently we use HermiT but it could be replaced by any other OWL reasoner. Our evaluation demonstrates a promising speedup for ontologies of different sizes and complexities that is linear to the numbers of cores.

II. Description Logics

We briefly introduce the core DL $\mathcal{ALC}$, which is a subset of OWL. We describe its syntax and semantics, selected inference services, and a tableau reasoning algorithm.

A. Syntax and Semantics

The Description Logic Attributive Concept Description Language ($\mathcal{ALC}$) proposed by Schmidt-Schauß and Smolka [16] was the first DL where a complete reasoning algorithm was provided. To formally define an $\mathcal{ALC}$ knowledge base, we denote with N_C a set of concept names of domain elements with common characteristics, N_R a set of role names representing a binary relationship between domain elements, and N_O a set of individual names within the represented domain.

The formal definition of the semantics of $\mathcal{ALC}$ is given by an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, consisting of a non-empty set $\Delta^{\mathcal{I}}$ called domain and an interpretation function $\cdot^{\mathcal{I}}$. The interpretation function $\cdot^{\mathcal{I}}$ maps every individual a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, every concept A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and

TABLE I
SYNTAX AND SEMANTICS OF DESCRIPTIONS IN $\mathcal{ALC}$

Syntax	Semantics
$\top$	$\Delta^{\mathcal{I}}$
$\bot$	$\emptyset$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$

every role R to a subset $R^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The description of syntax and semantics of $\mathcal{ALC}$ concept expressions is shown in Table I, where $C, D \in N_C$ are arbitrary concepts and $R \in N_R$ is a role.

Satisfiability: A concept C is satisfiable if there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$, i.e., there exists an individual $x \in \Delta^{\mathcal{I}}$ which is an instance of C, $x \in C^{\mathcal{I}}$. Otherwise, the concept C is unsatisfiable.

TBox: Terminological axioms include role inclusion axioms, which have the form $R \sqsubseteq S$ where $R, S \in N_R$, and general concept inclusion axioms (GCI), which have the form $B \sqsubseteq A$ where $A, B \in N_C$. A TBox consists of a finite set of terminological axioms. A TBox $\mathcal{T}$ is satisfiable if there exists an interpretation $\mathcal{I}$ that satisfies all the axioms in $\mathcal{T}$. The interpretation $\mathcal{I}$ is called a model of $\mathcal{T}$ and $\mathcal{T}$ is called consistent. For example, if the interpretation of a concept $B^{\mathcal{I}}$ is necessarily a subset of the interpretation of a concept $A^{\mathcal{I}}$ in all models of $\mathcal{O}$, then $\mathcal{O}$ entails $B \sqsubseteq A$ (abbr. $\mathcal{O} \models B \sqsubseteq A$). A concept equality definition of the form $C \equiv D$ is an abbreviation for the axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.

Subsumption: A concept D subsumes a concept C ($C \sqsubseteq D$) *iff* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$, i.e., every instance of C must be an instance of D. Subsumption can be reduced to satisfiability, i.e. subsumes$(D, C) \Leftrightarrow \neg sat(\neg D \sqcap C)$ and $C \sqsubseteq \bot \Leftrightarrow \neg sat(C)$.

Classification: The classification of a TBox results is in a subsumption hierarchy (or taxonomy) of all named concepts, with $\top$ as the root. If two named concepts A, B have a subsumption relationship, e.g., $A \sqsubseteq B$, then B is called an ancestor of A and A is a descendant of B. In case there exist no concepts A', B' such that $A \sqsubseteq B'$ and $B' \sqsubset B$ or $A \sqsubset A'$ and $A' \sqsubseteq B$, then B (A) is called a predecessor (successor) of A (B).

Additional Description Logic Constructors: $\mathcal{ALC}$ can be extended by various constructors that are denoted in the logic's name: $\mathcal{H}$ for role hierarchies, $+$ for transitive roles ($\mathcal{S}$ stands for $\mathcal{ALC}+$), $\mathcal{I}$ for inverse roles, $\mathcal{R}$ for role chain axioms ($\mathcal{R}$ includes $\mathcal{H}+$), O for nominals, Q for qualified number restrictions, $\mathcal{N}$ for number restrictions, and $(\mathcal{D})$ for using datatypes. For instance, OWL is a syntactic variant of the DL $\mathcal{SROIQ}(\mathcal{D})$ and $\mathcal{EL}$ is a subset of $\mathcal{ALC}$ supporting only $\sqcap$ and $\exists$.

B. Tableau Algorithm

A tableau algorithm decides the satisfiability of a given concept C by constructing a completion graph for C. A complete and clash-free completion graph for C is interpreted as C being satisfiable. A model is represented by a tableau completion graph where concept descriptions are built using boolean operators ($\sqcup$, $\sqcap$, $\neg$), universal restriction ($\forall$), and existential ($\exists$) value restriction on concepts. The tableau completion graph for $\mathcal{ALC}$ is a labeled graph $G = \langle V, E, L \rangle$, where each node $x \in V$ is labeled with a set $L(x)$ of concepts, and each edge $\langle x, y \rangle \in E$ is labeled with a set $L(\langle x, y \rangle)$ of roles. A completion graph G contains a clash, if $\{A, \neg A\} \subseteq L(x)$ for some atomic concept A, or $\bot \in L(x)$. The completion rules for $\mathcal{ALC}$ are shown in Table II. If no completion rule can be applied to the graph G, then it is complete. Example 2.1 illustrates how the tableau algorithm determines the satisfiability of $C = (A \sqcap \neg A) \sqcup B$.

TABLE II
THE COMPLETION RULES FOR $\mathcal{ALC}$

$\sqcup$-Rule	**If** $C \sqcup D \in L(v)$, for some $v \in V$ and $\{C, D\} \cap L(v) = \emptyset$, **then** choose $X \in \{C, D\}$ and X is added to $L(v)$
$\sqcap$-Rule	**If** $C \sqcap D \in L(v)$, for some $v \in V$ and $\{C, D\} \not\subseteq L(v)$, **then** C and D are added to $L(v)$
$\forall$-Rule	**If** $v, v' \in V$, v' is R-successor of v, $\forall R.C \in L(v)$ and $C \notin L(v')$, **then** C is added to $L(v')$
$\exists$-Rule	**If** $\exists R.C \in L(v)$, for some $v \in V$, and no R-successor v' of v such that $C \in L(v)$, **then** $v' \in V$, $L(\langle v, v' \rangle) = \{R\}$, C is added to $L(v')$

Example 2.1:
First we apply the tableau algorithm to:
$L(x) = \{C, (A \sqcap \neg A) \sqcup B\}$
The applicable rule is $\sqcup$-Rule. We apply it and obtain
$L(x) = \{A \sqcap \neg A, C, (A \sqcap \neg A) \sqcup B\}$
Then we can apply $\sqcap$-Rule and obtain
$L(x) = \{A, \neg A, A \sqcap \neg A, C, (A \sqcap \neg A) \sqcup B\}$
Finally, there is a clash between A and $\neg A$. Then we try the other disjunct and get
$L(x) = \{C, B, (A \sqcap \neg A) \sqcup B\}$
There are no more applicable rules and there is no clash. Therefore C is satisfiable and there exists a model $\mathcal{I}$ that satisfies $\Delta^{\mathcal{I}} = \{x\}$, $A^{\mathcal{I}} = \emptyset$, $B^{\mathcal{I}} = \{x\}$, $C^{\mathcal{I}} = \{x\}$.

III. PARALLEL TBOX CLASSIFICATION

Our goal is to parallelize the computation of subsumption taxonomies consisting of a large number of concepts and speed up the process of TBox classification. In order to reuse information from (non-)subsumption tests, our method implements a parallel framework and a shared-memory global data structure to record all binary subsumption relationships occurring in an ontology (or TBox) $\mathcal{O}$. A set P contains all *possible* subsumees that every concept could have and a set K represents all subsumees found from *known* subsumption relationships or subsumption tests. For example, if $\mathcal{O} \models B \sqsubseteq A$, then we insert B into K_A and delete B from P_A. Since the classification of $\mathcal{O}$ tests all pairs of concept subsumptions, we use the concepts remaining in possible subsumee sets to reflect the scale of work that still needs to be done until P becomes empty.

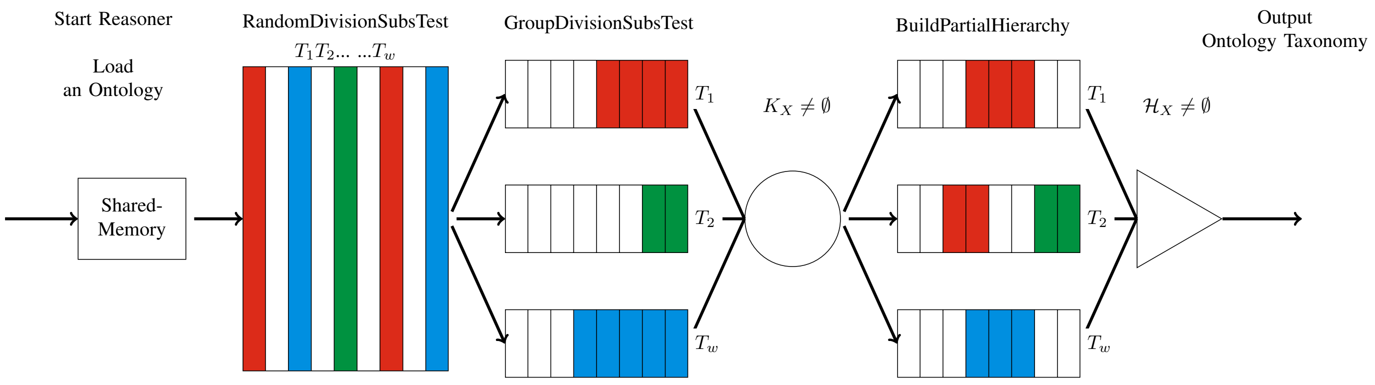


Fig. 1. The Architecture of Parallel TBox Classification Approach

After loading an ontology $\mathcal{O}$, a set $N_\mathcal{O}$ contains all concepts occurring in $\mathcal{O}$. For each concept $X \in N_\mathcal{O}$, our method initializes P_X, which contains all possible subsumees of X and an initially empty K_X to contain all the known subsumees derived from subsumption tests. For instance, let us assume three concepts $\{A, B, C\} \subseteq N_\mathcal{O}$. After having initialized P and K we get $P_A = \{B, C\}$, $P_B = \{A, C\}$, $P_C = \{A, B\}$ and $K_A = K_B = K_C = \emptyset$. Since $N_\mathcal{O}$ contains all concepts from $\mathcal{O}$, in the following phases we use $N_\mathcal{O}$ as a global parameter for classifying $\mathcal{O}$ in parallel. We use the predicate $subs?()$ to test subsumption relationships for each pair of concepts in P. The call of $subs?(B, A)$ returns true if A is subsumed by B and false otherwise. Before testing, it is necessary to know the satisfiability of each concept. A concept X is satisfiable if a model of $\mathcal{O}$ exists such that $X^\mathcal{I}$ is an non-empty set. Otherwise, the concept X is unsatisfiable. For example, if two concepts A, B are satisfiable, then the subsumption relationships of the pair $\langle A, B\rangle$ are tested using $subs?()$:

$$\{\langle \bot, B\rangle, \langle B, A\rangle, \langle A, \top\rangle\}$$

In addition, we use the set of $R_\mathcal{O}$ containing each concept $X \in N_\mathcal{O}$ where $P_X \neq \emptyset$.

The TBox classification process is implemented in three parallel phases. In each phase we use different parallelization strategies. As a global parameter w we specify the maximum number of parallel threads (or workers) available for classification. The architecture of our approach is shown in Figure 1. In the first phase, we randomly partition the set of all named concepts into disjoint sequences having almost identical sizes obtained by dividing the total number of named concepts by w. In the second phase, we find all concepts X with $P_X \neq \emptyset$ using a group division strategy with *round-robin scheduling* for the worker thread pool in order to finish the classification process. In the final phase, we implement a parallel *divide-and-conquer* framework. Partial hierarchies are generated in the divide part for all concepts X with $K_X \neq \emptyset$. In the conquer part the whole ontology is constructed based on the existing partial hierarchies where $\mathcal{H}_X \neq \emptyset$. The algorithm parallelTBoxClassification(P, K) is shown in Algorithm 1.

Algorithm 1: parallelTBoxClassification(P, K)

```
**Input:** $P, K$ - sets of *possible* and *known* subsumees
**Output:** $\mathcal{H}$ - the whole ontology taxonomy
$N_\mathcal{O} \leftarrow generateNodeSet(\mathcal{O})$
$T \leftarrow createWorkerPool()$
$L_\mathcal{O} \leftarrow getRandomOrder(N_\mathcal{O})$
$G \leftarrow randomDivision(L_\mathcal{O})$
**for each** *group* $G_i \in G$ **do**
    $T_i \leftarrow getAvailableThread(T)$
    $T_i \rightarrow randomDivisionSubsTest(G_i)$
$R_\mathcal{O} \leftarrow generateRemainingPossibleSet()$
$G \leftarrow groupDivision(R_\mathcal{O})$
**while** $R_\mathcal{O} \neq \emptyset$ **do**
    **for each** *group* $G_X \in G$ **do**
        $T_i \leftarrow getAvailableThread(T)$
        $T_i \rightarrow groupDivisionSubsTest(G_X)$
$X \leftarrow getTopConcept()$
**while** $K_X \neq \emptyset$ **do**
    $T_i \leftarrow getAvailableThread(T)$
    $\mathcal{H}_X \leftarrow (T_i \rightarrow buildPartialHierarchy(K_X))$
    **if** $\mathcal{H}_X \neq \emptyset$ **then**
        $\mathcal{H} \leftarrow buildOntologyTaxonomy(\mathcal{H}_X)$
    $X \leftarrow getKnownSubsumees(K_X)$
**return** $\mathcal{H}$
```

A. Ontology Classification

In the classification phase, we use two strategies, the random and the group division strategy. In our algorithm, we use for each concept global sets containing *possible* (P) and *known* subsumees (K). In that way we keep track of the changes caused by the pool of worker threads during classification. Each thread tests subsumption relationships and removes as many concepts from P as possible. TBox classification terminates once P has become empty for all concepts in $N_\mathcal{O}$.

Definition 1: With reference to $N_\mathcal{O}$, the set $R_\mathcal{O}$ contains all remaining possible subsumees of each concept and is defined in (1).

$$R_\mathcal{O} = \bigcup_{X \in N_\mathcal{O}} P_X \tag{1}$$

Algorithm 2: randomDivisionSubsTest(G_i)

```
Input: $G_i$ - random division group
Output: $K$ - sets of known subsumees
        $P$ - sets of remaining possible subsumees
for each concept pair $\langle X, Y \rangle \in G_i$ do
    if $\neg tested(X, Y)$ then
        $satX \leftarrow sat?(X)$
        $satY \leftarrow sat?(Y)$
        if $\neg satX$ then
            $P_X \leftarrow \emptyset$
            delete $X$ from $P_Y$
        else if $\neg satY$ then
            $P_Y \leftarrow \emptyset$
            delete $Y$ from $P_X$
        else
            if $subs?(X, Y)$ then
                insert $Y$ into $K_X$
            delete $Y$ from $P_X$
```

1) Random Division Strategy: According to the number of threads and total number of concepts occurring in $\mathcal{O}$, we divide all concepts into different groups with almost the same size. In order to make the best use of all idle threads, the number of threads is identical to the number of divided groups for testing subsumption relationships for all concepts in $N_{\mathcal{O}}$. Our method first generates an unordered sequence $L_{\mathcal{O}}$ which includes all concepts. Then we partition $L_{\mathcal{O}}$ into w different groups, where w is the number of available threads. Then we test subsumption relationships between all the pairs $\langle Y, X \rangle$ with $Y, X \in N_{\mathcal{O}}$ for each group G_i by calling randomDivisionSubsTest(G_i) (see Algorithm 2). We use $sat?()$ to test concept satisfiability and $tested()$ to check whether the subsumption between two concepts has already been tested.

Example 3.1: Assume there are three threads available to perform subsumption tests. The algorithm first shuffles all concepts in $N_{\mathcal{O}} = \{A, B, C, D, E, F\}$ and returns the first cycle sequence $L^1_{\mathcal{O}} = (A, C, E, D, B, F)$. Then each group G_i contains two possible subsumees, such as $G_1 = \{A, C\}$, $G_2 = \{E, D\}$, and $G_3 = \{B, F\}$ for subsumption testing. For each thread T_i the results are

$$T_1 : C \sqsubseteq A,\ T_2 : D \not\sqsubseteq E,\ T_3 : F \not\sqsubseteq B$$

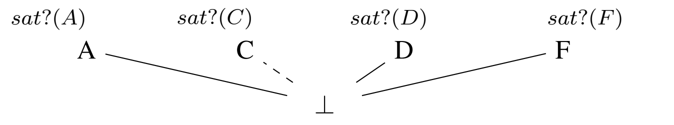
The second cycle sequence is $L^2_{\mathcal{O}} = (D, C, A, F, B, E)$. The divisions of each group are $G_1 = \{D, C\}$, $G_2 = \{A, F\}$ and $G_3 = \{B, E\}$. For each thread, the results are

$$T_1 : D \sqsubseteq C,\ T_2 : F \sqsubseteq A,\ T_3 : E \sqsubseteq B$$

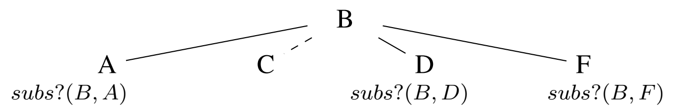
Since our process to generate random divisions currently ignores already discovered subsumptions, there is a possibility that a pair of concepts occurs in a division more than once in different cycles. Therefore, we use $tested()$ to avoid redundant tests. We consider the runtime for each thread as almost the same and the waiting time can be neglected right now. Currently, our results also show that the runtime differences for each thread can be neglected when compared with the total execution time.

Algorithm 3: groupDivisionSubsTest(G_X)

```
Input: $G_X$ - group division of concept $X$
Output: $K$ - sets of known subsumees
        $P$ - sets of remaining possible subsumees
for each concept $Y \in G_X$ do
    if $sat?(Y)$ and $\neg tested(X, Y)$ then
        if $subs?(X, Y)$ then
            insert $Y$ into $K_X$
        delete $Y$ from $P_X$
```



(a) Test satisfiability of the possible subsumees of B



(b) Test subsumptions for the possible subsumees of B

Fig. 2. Subsumption tests for group G_B

If $R_{\mathcal{O}}$ is not empty after random division phase testing, possible subsumees are left in P. We use a group division strategy to divide all remaining possible subsumees in $R_{\mathcal{O}}$ into different groups to continue testing subsumption relationships until P becomes empty.

2) Group Division Strategy: For each concept X in $N_{\mathcal{O}}$ a group $G_X = P_X$ is generated according to the remaining set $R_{\mathcal{O}}$ which is defined in Definition 1. The groups G_X define the input to groupDivisionSubsTest(G_X) (see Algorithm 3), which determines what elements of G_X are subsumed by X. Each group is assigned to a different idle thread until all groups have been classified. During the process, we apply *round-robin scheduling* to ensure a good use of all threads.

Example 3.2: According to the results from random division phase, let us assume the following six groups are generated:

$$\begin{array}{ll} G_A = \{B, D, E\} & G_B = \{A, C, D, F\} \\ G_C = \{B, E, F\} & G_D = \{A, B, F\} \\ G_E = \{A, C, F\} & G_F = \{B, C, D, E\} \end{array}$$

We further assume that three threads are available for subsumption testing. In Figure 2, we use G_B as an example. First the satisfiability of all concepts $Y \in G_B$ is tested using $sat?(Y)$. For example, we assume that C is unsatisfiable then this is shown in Figure 2(a) by a dashed line. For all satisfiable concepts $Y \in G_B$, we use $subs?(B, Y)$ to test subsumptions as shown in Figure 2(b).

TABLE III
SCHEDULING RESULTS OF EXAMPLE 3.2

	Timeline−−−−−−−−−−−−−−→	
T_1	G_A	G_E
T_2	G_B	G_F
T_3	G_C	G_D

In the following we now assume all concepts in all groups are satisfiable, the group scheduling is shown in Table III and the testing results of each thread are shown as follows.

$$\begin{aligned}
T_1(G_A) &: B \sqsubseteq A, D \sqsubseteq A, E \sqsubseteq A \\
T_2(G_B) &: A \not\sqsubseteq B, C \not\sqsubseteq B, D \not\sqsubseteq B, F \not\sqsubseteq B, A \equiv \top \\
T_3(G_C) &: B \not\sqsubseteq C, E \not\sqsubseteq C, F \sqsubseteq C \\
T_3'(G_D) &: A \not\sqsubseteq D, B \not\sqsubseteq D, F \not\sqsubseteq D \\
T_1'(G_E) &: A \not\sqsubseteq E, C \not\sqsubseteq E, F \not\sqsubseteq E \\
T_2'(G_F) &: B \not\sqsubseteq F, C \not\sqsubseteq F, D \not\sqsubseteq F, E \not\sqsubseteq F
\end{aligned}$$

Since P becomes empty and $R_{\mathcal{O}} = \emptyset$, all subsumption relationships between all concepts occurring in $\mathcal{O}$ have been tested. The classification of $\mathcal{O}$ terminates.

B. Ontology Taxonomy

In order to find the direct subsumees of each concept and build the whole subsumption hierarchy, we use a concept hierarchy strategy which is implemented by a parallel *divide-and-conquer* method to construct the taxonomy of $\mathcal{O}$. When $R_{\mathcal{O}}$ becomes empty, all known subsumees of a concept X are members of K_X. First we find the concept which is equal to $\top$ and traverse all the concepts $X \in K_\top$. Then we build the partial hierarchy $\mathcal{H}_\mathcal{X}$ for each concept X by computing the transitive closure to reduce the known set K_X. For each concept in K_X, we compute all the direct subsumees of X and insert them into $\mathcal{H}_\mathcal{X}$. Finally, the whole taxonomy of the ontology $\mathcal{O}$ is constructed based on the partial hierarchy of each concept.

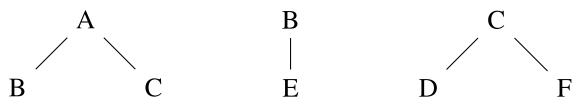
1) Concept Hierarchy Strategy: In the divide phase, the algorithm begins with K_X where X is initially equal to $\top$. For each concept $Y_i \in K_X$ and $i = 1, 2...n$, if $K_{Y_i} \neq \emptyset$ and $X \in K_{Y_i}$, then $Y_i \equiv X$; if $X \notin K_{Y_i}$, $Z_i \in K_{Y_i}$ and $Z_i \in K_X$, then Z_i is deleted from K_X. The method continues with the next concept $Y_{i+1} \in K_X$ until all the concepts in K_X have been traversed. The remaining concepts in K_X are the direct subsumees of X which are inserted into $\mathcal{H}_\mathcal{X}$. The algorithm buildPartialHierarchy(K_X) is shown in Algorithm 4. For each concept X with $K_X \neq \emptyset$ its partial hierarchy is built in parallel. The process terminates once all partial hierarchies have been built.

Example 3.3: According to the results from Example 3.2, when P becomes empty, the known sets for each concept are:

$$\begin{aligned}
K_A &= \{B, C, D, E, F\} \\
K_B &= \{E\},\ K_C = \{D, F\} \\
K_D &= \emptyset,\ K_E = \emptyset,\ K_F = \emptyset
\end{aligned}$$

Algorithm 4: buildPartialHierarchy(K_X)

```
Input: $K_X$ – set of known subsumees of concept $X$
Output: $\mathcal{H}_X$ – the partial hierarchy of concept $X$
if $K_X \neq \emptyset$ then
    for each concept $Y \in K_X$ do
        if $K_Y \neq \emptyset$ then
            if $X \in K_Y$ then
                delete $X$ from $K_Y$
                $setEquivalentConcept(X, Y)$
            else
                for each concept $Z \in K_Y$ do
                    if $Z \in K_X$ then
                        delete $Z$ from $K_X$
    $\mathcal{H}_X \leftarrow K_X$
return $\mathcal{H}_X$
```



(a) $\mathcal{H}_A$ (b) $\mathcal{H}_B$ (c) $\mathcal{H}_C$

Fig. 3. Partial Hierarchy for the concepts in different threads

Since $A \equiv \top$, the hierarchy construction starts with the first concept $B \in K_A$ and E is the first concept in K_B which is also in K_A, then E is deleted from K_A. The second concept is $C \in K_A$ and there are two concepts D and F in K_C, then D and F are deleted from K_A. Therefore $K_A = \{B, C\}$ and the partial hierarchy of A is $\mathcal{H}_A = \{B, C\}$. Since $K_B = \{E\}$ and $K_E = \emptyset$, the partial hierarchy of B is $\mathcal{H}_B = \{E\}$. Because of $K_C = \{D, F\}$, $K_D = \emptyset$ and $K_F = \emptyset$, the partial hierarchy of C is $\mathcal{H}_C = \{D, F\}$. The final partial hierarchy $\mathcal{H}$ of the concepts in each thread is as follows:

$$\begin{aligned}
T_1 &: \mathcal{H}_A = \{B, C\} \text{ in Figure 3(a)} \\
T_2 &: \mathcal{H}_B = \{E\} \text{ in Figure 3(b)} \\
T_3 &: \mathcal{H}_C = \{D, F\} \text{ in Figure 3(c)}
\end{aligned}$$

In the conquer phase, after the partial hierarchy of each concept has been built, we merge all the partial hierarchies into the whole taxonomy from top to bottom. The final concept hierarchy of $\mathcal{O}$ is shown in Figure 4.

IV. OPTIMIZATION

In order to minimize the potential non-possible or known subsumees from the *Possible* list, we first generates a unique index I for each concept occurring in $\mathcal{O}$. Each concept A with a smaller index I_A contains the possible relationships with concept B with a bigger index in P_A. Therefore the set P contains all possible relationships which could be possible subsumers or subsumees. In order to shrink the set P using less subsumption tests, we find that known results from subsumption tests can be used to prune untested possible concepts in P without subsumption testing. By using the results from Example 3.3, assume concept $B \in P_A$ will be tested for a subsumption relationship with A. The following

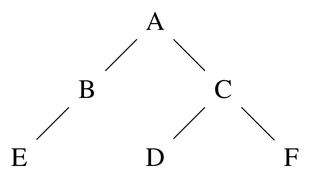


Fig. 4. The whole concept hierarchy of $\mathcal{O}$

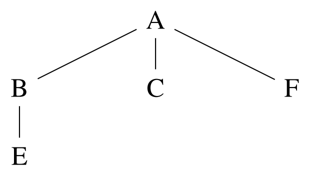


Fig. 5. An Example for Situation 2.3.1 and Situation 2.3.2

steps perform changes to P and K before new divisions are created for an idle thread.

Situation 1: If both concepts are unsatisfiable, their set P is empty; The changes to P and K are $P_A = \emptyset$, $P_B = \emptyset$, $K_A = \emptyset$ and $K_B = \emptyset$.

Situation 2: If both concepts are satisfiable, test the subsumption relationships between them.

Definition 2: Since $I_A < I_B$, the position of concept B in P_A is defined in (2).

$$B.position = P_A.position[I_B - I_A - 1] \quad (2)$$

Situation 2.1: If concept $B \in P_A$ and $tested(A, B)$ is true, which means B has been tested, then we continue with the next concept $C \in P_A$ to test its subsumption relationships with A; otherwise continue with Situation 2.2.

Situation 2.2: The subsumption relationships are tested in a symmetrical way by $subs?(B, A)$ and $subs?(A, B)$. If both results are true, then the two concepts are equivalent to each other; otherwise continue with Situation 2.3.

Situation 2.3: If only one of the results is true, i.e., $\mathcal{O} \models B \sqsubseteq A$ but $\mathcal{O} \models A \not\sqsubseteq B$, the changes to both sets P and K are P_A = {~~B~~, ~~C~~, D, E, ~~F~~}, $K_A = \{B, C, F\}$ and we continue with Situation 2.3.1; otherwise continue with Situation 2.4.

Situation 2.3.1: Delete all concepts $Y \in K_B$ from P_A and K_A. Due to $\mathcal{O} \models B \sqsubseteq A$ and $K_B = \{E\}$, all the subsumees of B are subsumees of A but not the direct subsumee of A as shown in Figure 5. Therefore, all concept $Y \in K_B$ are deleted from P_A without subsumption tests. In the example, concept $E \in K_B$ but $E \notin K_A$ is deleted from P_A. The changes of P are P_A = {~~B~~, ~~C~~, D, ~~E~~, ~~F~~} and we continue with Situation 2.3.2.

Situation 2.3.2: For all concepts $Y \in K_B$ delete A from P_Y. Due to $\mathcal{O} \models B \sqsubseteq A$ and $K_B = \{E\}$, all the subsumees of B are subsumees of A and concept A is not a subsumee of all concepts $Y \in K_B$ as shown in Figure 5. Therefore, concept A is deleted from P_Y with $Y \in K_B$. Since only concept $E \in Y$ and $I_E > I_A$ in our example, there are no changes to both P and K.

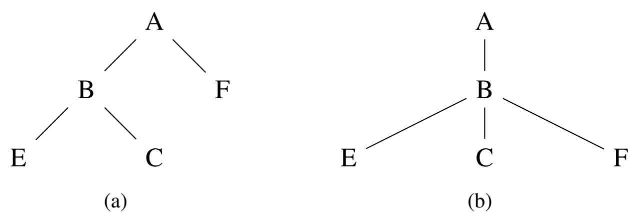


Fig. 6. Counter examples of 'delete all concepts $X \in K_A$ from P_B'

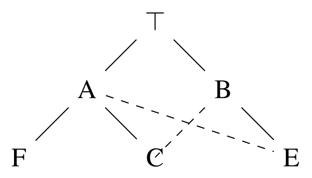


Fig. 7. First Counter Example for Situation 2.4

We also consider situations such as 'delete all concepts $X \in K_A$ from P_B'. Since $K_A = \{B, C, F\}$, we know that concepts C and F are in K_A and the two concepts could not have subsumption relationships with B. However, there are some counter examples which indicate possible relationships between B and C, F such that $\mathcal{O} \models C \sqsubseteq B$ in Figure 6(a) and $\mathcal{O} \models F \sqsubseteq B$ in Figure 6(b). Therefore, we cannot assume subsumption relationships between $\langle B, C\rangle$ and $\langle F, B\rangle$ without performing subsumption tests.

Situation 2.4: If both concepts are not subsumed by each other such that $\mathcal{O} \models A \not\sqsubseteq B$ and $\mathcal{O} \models B \not\sqsubseteq A$, then both sets P and K remain unchanged.

According to this condition, we try to find some situations which allow us to shrink P in an efficient way without performing subsumption tests. However, we identified some counter examples as shown in Figures 7+8 where the dashed lines indicate possible relationships between pairs of concepts. Below we describe two scenarios.

- Delete all concepts $X \in K_A$ from P_B and $Y \in K_B$ from P_A. For example as shown in Figure 7, there is a concept $C \in K_A$, $C \in P_B$, $\mathcal{O} \models A \not\sqsubseteq B$ and $\mathcal{O} \models B \not\sqsubseteq A$, but A and B are both known subsumers of C. The possible relationship between C and B could exist before C is deleted from P_B; there is a concept $E \in K_B$, $E \in P_A$, $\mathcal{O} \models A \not\sqsubseteq B$ and $\mathcal{O} \models B \not\sqsubseteq A$, but concept E is a subsumee of both A and B. Therefore, the relationships of the pairs $\langle B, C\rangle$ and $\langle A, E\rangle$ need to be tested before deleting C from P_B and E from P_A.
- For all concepts $X \in K_A$ delete B from P_X and all concepts $Y \in K_B$ delete A from P_Y. In the example shown in Figure 8(a), there is a concept $F \in K_A$, $F \in P_B$ ($I_F > I_B$), $\mathcal{O} \models A \not\sqsubseteq B$ and $\mathcal{O} \models B \not\sqsubseteq A$, but concept B is a known subsumee of F. In Figure 8(b), there is concept $E \in K_B$, $E \in P_B$, $\mathcal{O} \models A \not\sqsubseteq B$ and $\mathcal{O} \models B \not\sqsubseteq A$, but concept A is a subsumee of E. Therefore, relationships between the pairs of $\langle B, F\rangle$ and $\langle A, E\rangle$ need to be tested before deleting F from P_B

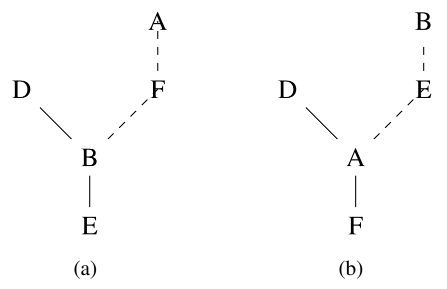


Fig. 8. More Counter Examples for Situation 2.4

$(I_B < I_F)$ and E from P_A $(I_A < I_E)$.

Algorithm 5 correctly deals with all the situations illustrated above.

Example 4.1: For random division tests, we apply the random division strategy and use the same random division results from Example 3.1. The first random division cycle result in:

$$T_1 : C \sqsubseteq A,\ A \not\sqsubseteq C$$
$$T_2 : E \not\sqsubseteq D,\ D \not\sqsubseteq E$$
$$T_3 : F \not\sqsubseteq B,\ B \not\sqsubseteq F$$

The results of the second random division cycle are:

$$T_1 : D \sqsubseteq C, C \not\sqsubseteq D$$
$$T_2 : F \sqsubseteq A, A \not\sqsubseteq F$$
$$T_3 : E \sqsubseteq B, B \not\sqsubseteq E$$

The remaining possible sets P are:

$$P_A = \{B, D, E\},\ P_B = \{C, D\},$$
$$P_C = \{E, F\},\ P_D = \{F\},\ P_E = \{F\}$$

For each random division cycle, we apply the above-mentioned optimized techniques. Since $\mathcal{O} \models C \sqsubseteq A$ and concept $D \in K_C$, concept D is deleted from P_A and the remaining sets P become:

$$P_A = \{B, E\},\ P_B = \{C, D\},$$
$$P_C = \{E, F\},\ P_D = \{F\},\ P_E = \{F\}$$

Now let us assume there are three threads available for subsumption testing and all concepts in $R_\mathcal{O}$ are divided into groups using the group division strategy. The divisions for the groups G_X are:

$$G_A = \{B, E\}, G_B = \{C, D\},$$
$$G_C = \{E, F\}, G_D = \{F\}, G_E = \{F\}$$

Algorithm 5: pruneNonPossible(A, B)

```
Input: $A, B$ - two concepts from $N_\mathcal{O}$
Output: $K$ - sets of known subsumees
        $P$ - sets of possible subsumees
if $sat?(A)$ then
    if $sat?(B)$ then
        if $\neg tested(B, A)$ and $\neg tested(A, B)$ then
            $result_1 \leftarrow subs?(A, B)$
            $result_2 \leftarrow subs?(B, A)$
            if $result_1$ and $result_2$ then
                return $A \equiv B$
            else if $result_1$ then
                for each concept $Y \in K_B$ do
                    delete $Y$ from $P_A$ and $K_A$
                    delete $A$ from $P_Y$
            else if $result_2$ then
                for each concept $X \in K_A$ do
                    delete $X$ from $P_B$ and $K_B$
                    delete $B$ from $P_X$
    else
        $P_B \leftarrow \emptyset$
else
    $P_A \leftarrow \emptyset$
```

After applying the optimized techniques and *round-robin scheduling* for each thread T_i, all the pairs in brackets have not been tested and the results are:

$$T_1 : B \sqsubseteq A,\ A \not\sqsubseteq B\ (E \sqsubseteq A,\ A \not\sqsubseteq E);$$
$$T_2 : B \not\sqsubseteq C,\ C \not\sqsubseteq B\ (B \not\sqsubseteq D,\ D \not\sqsubseteq B);$$
$$T_3 : (E \not\sqsubseteq C,\ C \not\sqsubseteq E)\ F \sqsubseteq C,\ C \not\sqsubseteq F;$$
$$T_1' : D \not\sqsubseteq F,\ F \not\sqsubseteq D;$$
$$T_2' : (E \not\sqsubseteq F,\ F \not\sqsubseteq E)$$

For T_1, the subsumption relationship between concepts A and B is that $\mathcal{O} \models B \sqsubseteq A$, then concept $E \in K_B$ can deleted from P_A without further testing by applying Situation 2.3.1. For T_2, the concepts B and C are not subsumed by each other and $D \in K_C$, then concept D can be deleted from P_B without further tests. For T_3, since the concepts B and C are not subsumed by each other and $E \in K_B$, concept E can be deleted from P_C without further tests. Since we use a global atomic data structure when testing the relationships between B and C, there will be no conflict between T_2 and T_3. The subsumption tests between C and E can be executed only after concepts B and C have been tested. For T_2', since concept $E \in K_B$, $F \in K_C$ and concepts B and C are not subsumed by each other, F can be deleted from P_E without further tests.

Therefore, all the subsumptions listed in brackets can be inferred without testing. The remaining possible set $R_\mathcal{O}$ will be pruned significantly due to the many relationships found among the concepts. The classification terminates when P has been emptied.

TABLE IV
METRICS OF USED OWL TEST ONTOLOGIES

Ontology	Concept	Axiom	SubClassOf	Expressivity
WBbt.obo	6785	19138	12347	$\mathcal{EL}$
EHDA#EHDA	8341	33367	8339	$\mathcal{EL}$
obo.PREVIOUS	1663	4099	1377	$\mathcal{ELH}+$
actpathway.obo	7911	25314	17402	$\mathcal{EL}$
EHDAA2	2726	16818	13458	$\mathcal{ELH}+$
lanogaster.obo	10925	16567	5641	$\mathcal{EL}$
MIRO#MIRO	4366	21274	4454	$\mathcal{EL}+$
CLEMAPA	5946	16864	10916	$\mathcal{EL}$
EMAP#EMAP	13735	27467	13732	$\mathcal{EL}$

V. A FIRST EVALUATION

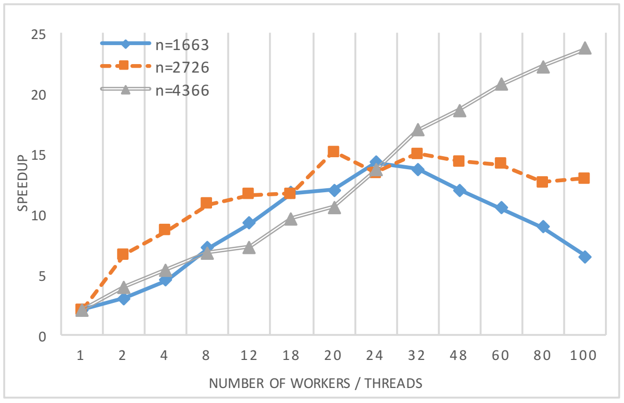
Our parallel classification architecture is implemented as a Java shared-memory program using HermiT 1.3.8 as OWL reasoner plug-in. We performed our experiments on a HP DL580 Scientific Linux[1] SMP server with four 15-core processors (Gen8 Intel Xeon E7-4890v2 2.8GHz) and 1 TB RAM. For this first evaluation of our classification architecture we selected a set of 9 real-world ontologies from the ORE 2015 [17] repository that contain up to 13,000 concepts and 33,000 axioms to test scalability and 5 ontologies from the ORE 2014 [18] repository that contain up to 7,000 axioms and 967 qualified cardinality restrictions (QCRs), which are used to constrain the number of values of a particular property and type and are considered to be an important parameter in testing the complexity factors of our approach. Their metrics are shown in Tables IV+V (see Section II-A about naming DLs). For benchmarking we ensured exclusive access to the server in order to avoid that other jobs affect the elapsed time of our tests. For tests with a smaller set of threads we ran several jobs in parallel but jobs exceeding 60 threads were run exclusively. Currently, since our method is implemented using an independent parallel synchronized architecture for all the processing threads and the overhead factors mainly focus on the scalability and complexity of the ontologies we chose.

A. *Ontology Scalability*

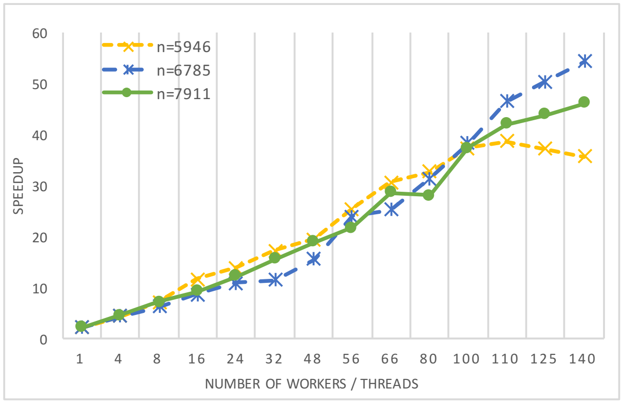
In order to assess the scalability of our architecture we conducted a series of experiments where the number of workers/threads available for classification varied between 1 (sequential case) to 140. Due to the limitations of our test environment we restricted the maximum number of threads to 140. We computed the speedup as the ratio of the runtime (sum of runtimes of all threads) divided by the elapsed time. Each individual experiment was repeated three times and the resulting average was used to determine its runtime and elapsed time. The 9 ontologies can be roughly divided into three groups of similar sizes measured by their number of contained concepts (n).

Figure 9(a) shows a set of smaller ontologies. For the two smallest ontologies the peak speedup is reached with 20-32 workers. A higher number of workers indicates a performance degradation that is due to our current partitioning scheme

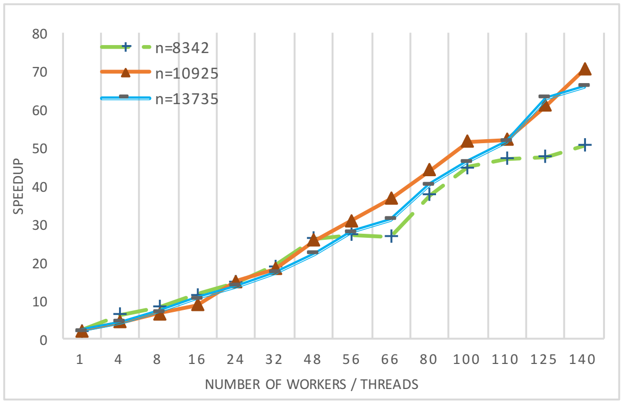
[1]GNU/Linux Version 2.6.32-642.15.1.el6.x86_64



(a) $n \in \{1663, 2726, 4366\}$



(b) $n \in \{5946, 6785, 7911\}$



(c) $n \in \{8341, 10925, 13735\}$

Fig. 9. Speedup factors for ontologies from Table IV with an increasing number of concepts (n = number of concepts)

where the size of the partition allocated to of each worker is roughly $\frac{n}{w}$ (n is the number of concepts in an ontology and w the number of workers). When the partition size becomes too small, overhead affects the performance adversely.

Figure 9(b) shows medium-sized and Figure 9(c) large ontologies. With the exception of the smallest ontology in Figure 9(b) both figures show a similar speedup increase. This is due to bigger partition sizes and reduced overhead. The peak is currently reached with 140 workers. It is necessary to make partition sizes reasonably big. Therefore, for our future

TABLE V
METRICS OF USED OWL TEST ONTOLOGIES WITH QCRS

Ontology	Concept	Axiom	SubClassOf	#QCRs	#Somes	#Alls	Equivalent	Disjoint	Expressivity
ncitations_functional	2332	7304	2786	47	659	54	269	115	$\mathcal{SROIQ(D)}$
nskisimple_functional	1737	4775	2234	43	533	27	50	84	$\mathcal{SRIQ(D)}$
rnao_functional	731	2884	1235	446	774	2	385	61	$\mathcal{SRIQ}$
ddiv2_functional	1469	4080	1832	48	388	27	56	75	$\mathcal{SRIQ(D)}$
bridg.biomedical_domain	320	6347	295	967	0	0	5	37	$\mathcal{SROIN(D)}$

research we are expecting a similarly good or even better performance for much bigger ontologies (number of concepts up to 300,000).

B. Ontology Complexity

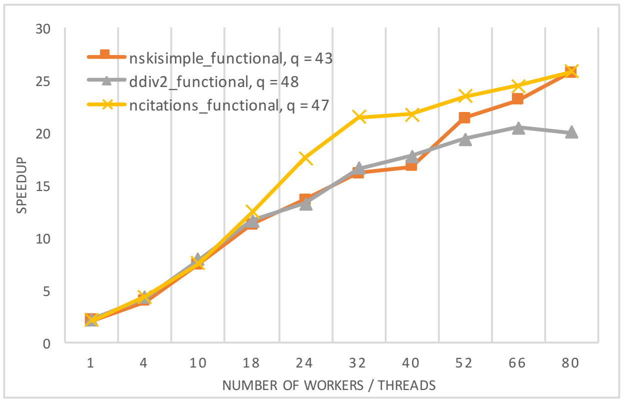
There are other factors that can affect our experiments such as the complexity of an ontology and the efficiency of HermiT, the selected reasoner plug-in, which is also implemented in Java. For most of the used ontologies we observed that the runtimes of individual subsumption tests performed by HermiT are rather uniform but for ontologies with a higher expressivity it is well known that just a few subsumption tests may require a significant amount of the total runtime. Furthermore, the plug-in reasoner might be more or less efficient depending on the expressivity of the test ontologies.

In order to test the performance of our architecture for complex ontologies, we used the same experimental environment and selected five smaller real-world ontologies with a logic of high expressivity as shown in Table V. Since the number of concepts for these ontologies is only up to 2332, we conducted experiments where the number of available workers range from 1 to 80. We computed the speedup as the ratio of runtime divided by elapsed time. Each experiment was repeated three times and used the resulting average as runtime and elapsed time to calculate speedup. We roughly divided the five ontologies into two groups based on their number of QCRs.

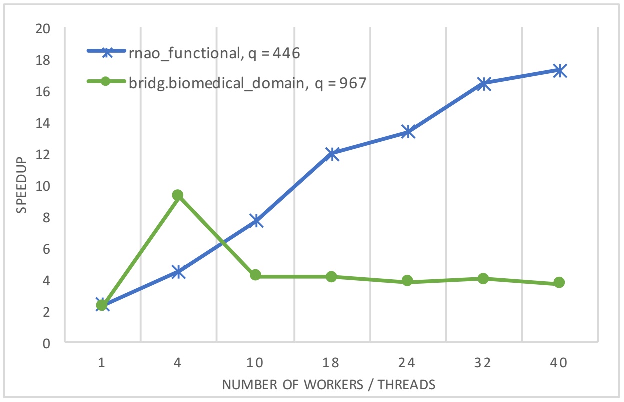
In Figure 10(a), the number of QCRs in the first group is around 40. Since we try to select reasonable partition sizes, we used up to 80 threads to compute the speedup factors for all three ontologies. As the number of threads is increased, a better speedup is observed.

In Figure 10(b), the number of QCRs is reaching 967, which indicates the difficulty of ontology classification. With an increasing number of threads, the ontology with 446 QCRs shows a good speedup factor. However, due to the complexity and limitation of HermiT, the other one which has 967 QCRs shows the best performance for four workers and afterwards the speedup factor remains around 4. As we observed, this ontology includes some difficult QCRs which causes several subsumption tests to take longer than others, therefore its speedup does not always increase.

As expected, in general the results show that our method has a speedup linear to the number of threads. However, since our algorithm currently is implemented without enhanced optimizations to reduce the number of subsumption tests (besides the ones we described in Section IV), we are adding various optimizations to better compete with existing sequential reasoners.



(a) $n \in \{1469, 1737, 2332\}$, $q \in \{43, 47, 48\}$



(b) $n \in \{320, 731\}$, $q \in \{446, 967\}$

Fig. 10. Speedup factors for ontologies from Table V with QCRs (n = number of concepts, q = number of QCRs)

C. Load Balancing

From our experiments we observed that the first (random division) phase (with randomly created groups of similar average size) exhibits a better load balancing than the second (group division) phase. However, the classification process can only terminate once the second phase has been completed. To get a better understanding of the performance for both the random division and the group division phase and balance the two phases to make them more efficient, we used a ratio representing the decrease of the number of possible subsumers in each phase.

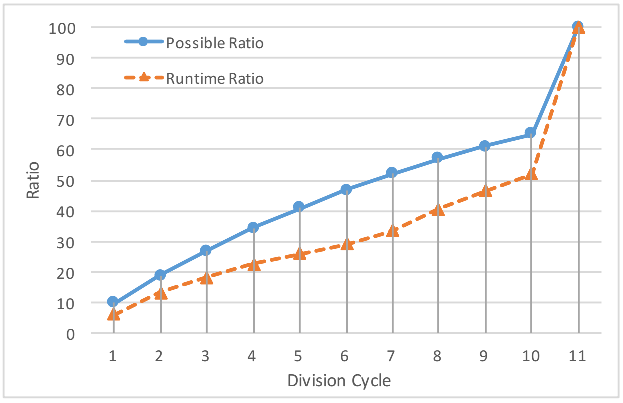


Fig. 11. Division cycle result of *ncitations_functional.owl* (concepts = 2332, threads = 10, random division cycle = 10, group division cycle = 1)

Definition 3: *InitialPossible* is defined as the initial number of possible subsumers for an ontology and *Remaining-Possible* is the number of possible subsumers after completing each division cycle. Therefore, the possible ratio is defined in (3) as follows.

$$Possible = \frac{InitialPossible - RemainingPossible}{InitialPossible} \tag{3}$$

We chose the ontology *ncitations_functional.owl* from Table V with 2332 concepts and used 10 workers. We decided on ten random division cycles and one group division cycle to determine the load balance factors. We also recorded the runtime for each phase and calculated the runtime ratio as the accumulated cycle runtimes divided by the total runtime. The result is shown in Figure 11.

We implemented two parallel classification phases in our methods and the random division phase applied a completely random division strategy to minimize the number of remaining possible subsumers on a large scale. As expected, the random division strategy (cycles 1-10) increased the value *Possible* up to 60, i.e., the number of possible subsumees was reduced by 60%, before the group division strategy was applied. The runtime ratio is almost at the same level as the possible ratio (see Figure 11). However, from our test results we noticed that with an increasing number of threads, the ratio factor does not necessarily increase, especially if the number of threads is more than 60. We are still working on finding a better load balancing between the two phases which can both shorten the runtime and reduce the number of possible subsumees as quickly as possible. Therefore, the ratio factor affecting load balancing of the two parallel phases can be expected to be improved when much larger ontologies are tested.

VI. Conclusion

We have presented promising results in designing and implementing a parallel OWL ontology classification architecture. We were inspired by ideas from [15] and applied parallel techniques to create a thread pool for each sub-task working on an independent processor. Compared to existing sequential classification methods and the limitations of recently proposed parallel classification approaches, our method is the first using a random division strategy to address the large scale of ontologies and then applying a group division strategy to finish TBox classification. Furthermore, due to the design of our shared atomic data structures we avoid possible race conditions for updates of shared data. Currently, our method relies on the existing sequential OWL reasoner HermiT. We observed that for difficult ontologies our method can outperform the stand-alone version of HermiT. However, due to processor and reasoner restrictions, not all ontologies could be tested on the current platform within a reasonable amount of time. More thorough experiments on the speedup and ratio factors of our parallel framework are in progress.

From our current results, we believe that we can apply our approach to large scale ontologies to get a better performance compared to existing sequential methods. We plan to achieve a better load balance solution for the two parallel phases. Overall, our parallel TBox classification method shows promising results that makes us believe it could be applied to ontologies with a much larger size and complicated ontologies with a promising runtime and a better speedup.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila, “The semantic web,” *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001.
[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook*, 2nd ed. Cambridge University Press, 2007.
[3] R. Möller and V. Haarslev, *Tableau-Based Reasoning*. Handbook on Ontologies-Tableau-based reasoning, 2009, pp. 509–528.
[4] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz, *OWL 2 Web Ontology Language Profiles (Second Edition)*. W3C Recommendation, 2009.
[5] V. Haarslev and R. Möller, “RACER system description,” in *International Joint Conference on Automated Reasoning*, 2001.
[6] D. Tsarkov and I. Horrocks, “Fact++ description logic reasoner: System description,” in *International Joint Conference on Automated Reasoning*, 2006.
[7] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, “HermiT: an OWL 2 reasoner,” *Journal of Automated Reasoning*, vol. 53, no. 3, pp. 245–269, 2014.
[8] A. Steigmiller, T. Liebig, and B. Glimm, “Konclude: system description,” *Web Semantics*, vol. 27, pp. 78–85, 2014.
[9] Y. Kazakov, M. Krötzsch, and F. Simančík, “Concurrent classification of $\mathcal{EL}$ ontologies,” in *Int. Semantic Web Conf.* , vol. 305-320. Springer, 2011.
[10] M. Aslani and V. Haarslev, “Parallel tbox classification in description logics - first experimental results,” in *Proc. of the 19th European Conf. on Artificial Intelligence*, 2010, pp. 485–490.
[11] K. Wu and V. Haarslev, “A parallel reasoner for the description logic $\mathcal{ALC}$,” in *Proc. of the 2012 Int. Workshop on Description Logics*, 2012, pp. 378–388.
[12] ——, “Exploring parallelization of conjunctive branches in tableau-based description logic reasoning,” in *Proc. of the 2013 Int. Workshop on Description Logics (DL-2013)*, 2013, pp. 1011–1023.
[13] ——, “Parallel OWL reasoning: Merge classification,” in *Proc. of the 3rd Joint Int. Semantic Technology Conference*, 2013, pp. 211–227.
[14] J. Faddoul and W. MacCaull, “Handling non-determinism with description logics using a fork/join approach,” *International Journal of Networking and Computing*, vol. 5, no. 1, pp. 61–85, 2015.
[15] B. Glimm, I. Horrocks, B. Motik, R. Shearer, and G. Stoilos, “A novel approach to ontology classification,” *Web Semantics*, vol. 14, pp. 84–101, 2012.
[16] F. Baader and U. Sattler, “An overview of tableau algorithms for description logics,” *Studia Logica*, vol. 69, pp. 5–40, 2001.
[17] 4th OWL reasoner evaluation (ORE) workshop, 2015.
[18] 3rd OWL reasoner evaluation (ORE) workshop, 2014.