A Parallel Pairwise-Clustering Matching Algorithm for Large-Scale Metadata Models Using Levenshtein Distance

Seham MOAWED[1]*, Ali ELDOSOUKY[1], Amany SARHAN[2], Sally ELGHAMRAWY[3]

[1] Department of Computer Engineering, Mansoura University, Mansoura, Egypt
schemamatching2020@gmail.com, schema@std.mans.edu.eg (**Corresponding author*)

[2] Department of Computer Engineering, Tanta University, Tanta, Egypt
am_sarhan_ya@yahoo.com

[3] Computers Engineering Department, MISR Higher Institute for Engineering and Technology, Mansoura, Egypt
sally_elghamrawy@ieee.org

Abstract: Data integration is required for applications processing multiple data sources; consequently, semantic heterogeneity has become increasingly severe. The problem has been faced inevitably, imposing the need for metadata models matching to discover correspondences across different metadata models. However, in the large-scale scene, current metadata model matching systems suffer from memory consumption and a lack of scalability. Partitioning and parallelized techniques have been proposed to reduce space and temporal complexities. Nevertheless, few studies have been conducted to ameliorate matching efficiency on Graphical Processing Units (GPUs) when clustering techniques are utilized. To this end, the present paper proposes an efficient hardware implementable matching algorithm on GPU dedicated to the large-scale metadata models, named PPM (Parallel Pairwise Matching), which depends heavily on approximate string matching using the k-difference (ASM). In PPM, parallel processing is based on row flow computing in a way that eliminates data dependency obstacles exposed to the matching space. In addition, it decreases the amount of data transferred over the GPU. At most, one individual matching task will simultaneously be assigned to the device for parallel manipulation. This overcomes bringing out all independent matching tasks into bulks to the device where individual matching tasks can be separated and carried out in parallel. Extensive tests utilizing various workloads of metadata models on a CUDA-enabled GPU of NVIDIA GeForce GTX 860M have demonstrated the validity of the present assertions. The results imply that the proposed algorithm outperforms the sequential algorithm by up to 5.21-30.70 times.

Keywords: Metadata models, Large-scale matching, Partitioning-based matching, Scalability, Parallel computing.

1. Introduction

Today, the growth of metadata models in the scientific and commercial domains has exploded. Metadata models can be described as domain specifications with the main goal of specifying a metadata language (Coyle, 2010). A metadata model can be an XML schema and/or an ontology (Jung et al., 2012; Sivasankari & Shomona, 2016). The World Wide Web Consortium (W3C) Semantic Web Activity is a continuing endeavor to facilitate metadata model integration and sharing among many applications and parties. Unfortunately, WEB is confronted with databases containing massive amounts of data in various representations. The issue of managing heterogeneity among various information resources is becoming more prevalent. Metadata model matching (Mani & Annadurai, 2021; Ochieng & Kyanda, 2018) handles semantic heterogeneity. It discovers correspondences between entities in semantically relevant metadata models. Many application domains rely on the matching operation, including the semantic web, data warehouse, ontology integration, e-commerce, sensor networks, peer-to-peer systems, semantic web services, and social networks. However, challenges of metadata models matching problem exist in two primary categories: (i) accuracy, which measures the effectiveness of the matching process; and (ii) performance, which measures the time required for the matching process to execute (Sellami et al., 2008).

Although semantic web researchers have made considerable efforts to enhance the accuracy of metadata model matching systems, the performance of large-scale matching remains a concern. Several obstacles arise when matching large metadata models. First, matching metadata models is a computationally demanding process with quadratic computational complexity. Second, distinct element measures (matchers) should be used to obtain high matching quality. When matchers are applied serially, the matching task takes a long time. Third, there is a trade-off between accuracy and execution speed. As a result, to be suited for matching large metadata models, a tool must contain strategies for dealing with the complex matching process and decreasing search space and time computations (Babalou et al., 2016; Ochieng & Kyanda, 2018).

Approximate string matching (ASM), such as Levenshtein Distance (Navarro, 2001) is one of the resilient string metrics that has been successfully utilized in numerous applications. It supports the three most popular edit operations: insertion, deletion, and substitution. A dynamic solution for the k-difference ASM problem has been released to enhance the standard sequential algorithm. It is distinguished by its widespread usage in metadata model matching (Chong & Lee, 2022; Xue et al., 2021) and broad applicability to different hardware aspects, GPUs being among them (Chen et al., 2017; Tran et al., 2016).

Parallel processing with Graphical Processing Units (GPUs) has recently attracted a lot of attention as an intriguing alternative to standard microprocessors in high-performance computer systems (HPCs) (Owens et al., 2008). The emergence of high-performance matching approaches in terms of execution time dedicated to large-scale metadata models gives rise to harvesting GPU computational power capabilities in the realm of metadata model matching.

To this end, the research is based on an efficient hardware-implementable matching algorithm, which combines two promising areas, partition-based and parallel matching strategies. The algorithm enhances large-scale metadata models matching scalability in terms of speed on GPUs, in the presence of clustering methods. It mainly parallelizes dynamic approximate string matching. It relies on an agglomerative hierarchical algorithm (Algergawy et al., 2011), with a bottom-up clustering approach to divide the parsed model tree into divisions based on context-based structural node similarities. The efficiency of the matcher lies in the size of data remaining inside the GPU. Instead of accumulating all pairs of the similarity set that emerged from the similar clusters detecting stage into bulks on GPU, where individual pairs are then split up to occupy available cores, only a single pair of the similarity set is transferred to the device at a time for parallel computing. Furthermore, PPM can parallelize the calculation of elements in the same row of the edit distance matrix, removing data dependency.

The merit of the algorithm proposed in this paper is demonstrated through experiments on varying-sized datasets; some of them have been widely used in various matching prototypes in the Ontology Alignment Evaluation Initiative (OAEI) campaign (Ontology Alignment Evaluation Initiative, n. d.). The experimental findings revealed that the suggested algorithm outperformed the sequential algorithm by up to 5.21- 30.70 times. The main contributions of the paper are the following:

- It focuses on the problem of executing clustering on GPUs and finding the best solution;
- It presents a fast and scalable matching algorithm to deal with clusters and perform fast mapping computing for large-scale metadata models on GPU, namely, PPM;
- It links Java programming language to CUDA kernels with the JCUDA interface;
- It conducts some experiments on datasets with varying sizes to demonstrate how the proposed matching algorithm improves matching efficiency.

The following is the outline for the paper. Section 2 overviews the previous methods for partitioning and parallelizing the large-scale metadata models. The overall design of the proposed framework is depicted in Section 3. Section 4 discusses the CUDA implementation of the proposed parallel matcher. The experimental results from the most commonly used datasets follow in Section 5. Finally, Section 6 summarizes the main findings and discusses future directions.

2. Related Work

There have been many tools developed for large-scale metadata model matching (Otero-Cerdeira et al., 2015). However, as the popularity of metadata models has grown, so have the challenges that metadata model matching tools must overcome in order to establish high-quality correspondences between metadata models, while working with limited computing resources (Shvaiko & Euzenat, 2008). As a result, supplemental techniques beyond those required for matching medium- and small-sized metadata models are necessary for a tool to take on the challenge of matching such large metadata models. This section will cover techniques for matching large metadata models. Figure 1 demonstrates the hierarchy of scalability techniques for large-scale metadata models.

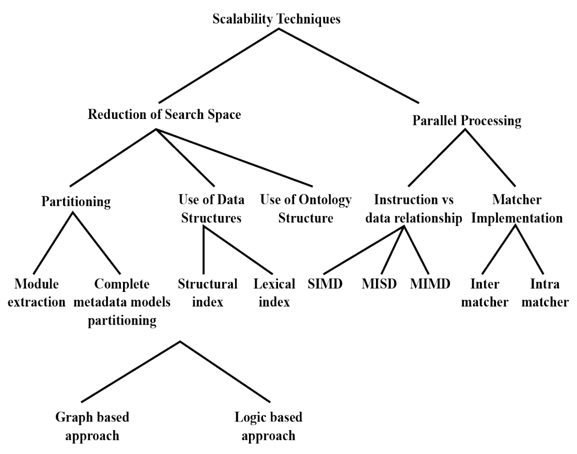

Figure 1. Hierarchy of Scalability Techniques (Ochieng & Kyanda, 2018)

2.1 Scalability Techniques

Scalability strategies utilized by metadata models matching tools are explored to alleviate the high time and space complexity involved with matching large metadata models, in order to thoroughly assess the state-of-the-art methodologies. Tools can be categorized into two types (Rahm, 2011), reduction of search space techniques and parallel matching.

2.1.1 Reduction of Search Space

To decrease the search space, metadata model matching technologies currently use three techniques: partitioning, data structure usage (El Abdi et al., 2015; Ngo & Bellahsene, 2016), and metadata model structure use (Huber et al., 2011; Wang, 2010).

2.1.2 Partitioning

To facilitate the matching process, metadata model matching technologies employ two forms of partitioning: module extraction and complete metadata models partitioning (Ochieng & Kyanda, 2018). In module extraction, extracting a module from a fragment and reasoning over the module rather than the whole metadata model can substantially speed up the reasoning process and optimize memory usage. For complete metadata model partitioning methodologies, a large metadata model is broken into smaller divisions based on preset criteria. Complete metadata model partitioning improves scalability and shrinks the search space by supporting parallelization, maintaining the effectiveness of a matching tool, and reducing time complexity.

According to (Abadi & Zamanifar, 2011), there are two types of metadata model partitioning methods: Graph-based Approach and Logic-based Approach. Graph-based Approach uses graph-based techniques to traverse the metadata model hierarchy to extract partitions. Because it avoids the reasoning approach, the strategy is scalable, but it may result in insufficient partitions, because it ignores the semantics modelled in the underlying metadata model language. The articles (Kusnierczyk, 2008; Schlicht & Stuckenschmidt, 2007) provide graph-based metadata model partitioning algorithms. Logic-based Approach partitions a metadata model utilizing description logic. It builds more extensive partitions than the Graph-based Approach, by using the relationships specified in the metadata model. It is less scalable than the Graph-based Approach, since it is reliant on reasoning. This approach is used in several works, as in (Cuenca Grau et al., 2007; Cuneca Grau et al., 2008). Partitioning is used by several metadata model-matching tools to reduce time complexity (Groß et al., 2012; Saruladha & Ranjini, 2016).

2.1.3 Parallel Composition

Parallel matching techniques are primarily advocated to reduce ontology matching execution time by distributing concept comparisons across the resources of a distributed system (Groß et al., 2010). Ontology matching tool parallelization can be examined in two ways: instruction vs. data relationship and matcher implementation.

2.1.4 Instruction vs. Data Relationship

This category divides parallelization implementations based on whether the same instruction runs on various datasets or multiple instructions are executed on the same or different datasets. In this context, three basic parallelization strategies are used (Tenschert et al., 2009): single instruction multiple data (SIMD), multiple instruction single data (MISD), and multiple instruction multiple data (MIMD). In the SIMD paradigm, all processing units (PUs) execute the same instruction on different data elements. This approach has been implemented in ontology matching by several tools at various stages of the ontology matching process, as in (Shvaiko et al., 2016). In MISD parallelism implementation, each PU executes a distinct instruction on the same dataset. XMAP++ (Djeddi et al., 2014) and MaasMatch (Schadd & Roos, 2014) are some tools that implement MISD. MIMD is a parallelism implementation in which each PU executes a distinct instruction on a different independent dataset. SPHeRe (Amin et al., 2014) implements MIMD information of data parallelism during the entity-matching stage of the ontology matching process.

2.1.5 Matcher Implementation

Parallel matching algorithms are divided into inter- and intra-matcher parallelization (Kirsten et al., 2011). Inter-matcher matching parallelizes the execution of independently executable parallel matchers. The number of independent matchers limits this type. Intra-matcher divides the metadata models required for a particular use case into multiple parts. Each part of the portioned metadata model is matched with the concepts of another one in a distributed manner and/or in parallel. In addition, intra-matcher parallelism can be applied to sequential and independently executable matchers, allowing it to be combined with inter-matcher parallelism. This approach has been implemented in (Groß et al., 2012). Several tools have been presented to implement parallel matching (Amin et al., 2016; Kirsten et al., 2011).

GOMMA (Groß et al., 2012; Kirsten et al., 2011) is a tool that allows for extremely parallel string matching on Graphical Processing Units (GPUs). It optimizes n-gram matching for determining the similarity of concept names and synonyms, and it supports GPU-based parallel matching. To this end, another type of string-matching techniques is parallelized on GPU, namely the Levenshtein distance, commonly used throughout the metadata models matching field (Chong & Lee, 2022; Xue et al., 2021) to allow efficient execution for matching.

3. The Proposed Matching Framework

This section presents an overview of the proposed high-performance GPU matching framework dedicated to large-scale metadata models. The matching framework, as shown in Figure 2, involves a set of modules, including parsing, clustering, similar cluster determining and data preparation, matching, and alignment evaluation modules. First, the benefit of each constituting module is summarized, followed by a thorough explanation of how the matching module is implemented. The matching framework is distinguished by its ability to match schemas and ontologies.

It includes the XSOM parser for schemas (Java Enterprise Edition, n d.) and the Jena API parser (Apache Jena, n. d.) for ontologies. The input metadata models should be depicted internally using a common data model in order to build a generic matching framework. The data model should be capable of normalizing metadata models represented by diverse metadata model languages, reducing syntax inconsistencies across metadata models. Graphs are used as an internal representation for matching metadata models. There are several reasons for selecting graph as an internal representation for the metadata models to be matched. First, graphs are well-known data structures and have their algorithms and implementations. Second, the metadata model matching problem is turned into another standard problem, namely, the graph matching, by using the graph as a common data model (Do & Rahm, 2007; Zhang et al., 2006). Using a set of predetermined transformation rules similar to those in (Lee et al., 2002), a graph can be turned into a tree representation by dealing with nesting and repetition problems. The graph contains a finite number of nodes and edges. An object identifier uniquely identifies each node, which expresses component attributes such as element name, data type, and constraint. On the other hand, each edge reflects the relationship between every two nodes.

As explained in (Algergawy et al., 2011), a hierarchical decomposition clustering approach

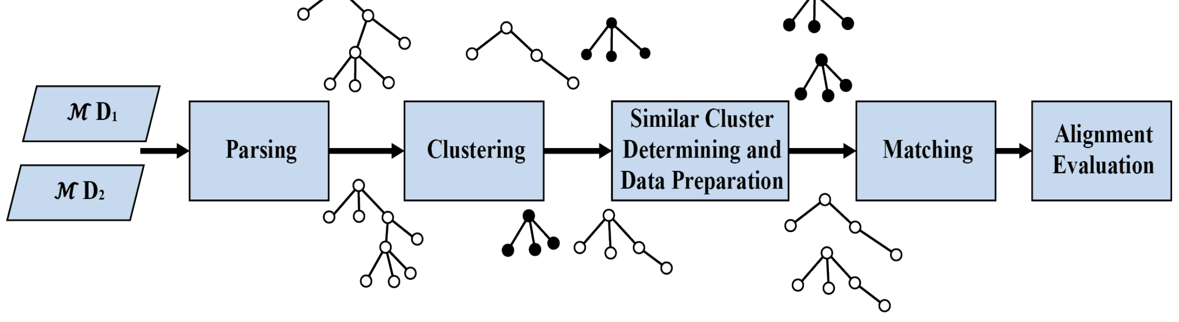

Figure 2. Proposed matching framework

with an agglomerative nature is used. It builds a tree depicting the cluster hierarchy in a bottom-up way. It computes the structural similarity between every pair of nodes, υi and υj, using the concept of node context. The node context is described as the surroundings of the node, including the node itself and all the parents and children of the node. For detecting the most similar clusters, the framework is adopted with two methods: Vector Space Model (VSM) and Latent Semantic Indexing (LSI) (Algergawy et al., 2014). This step reduces matching overhead by excluding dissimilar partition pairs from further matching processing.

To fully use the huge number of CUDA cores, the data reproduced from the clustering stage has to be well organized. Consider two metadata models, $\mathcal{MD}_1$, $\mathcal{MD}_2$, which are partitioned into two sets of clusters $C_{set1} = [1, \ldots, m]$, $C_{set2} = [1, \ldots, n]$, respectively. A similarity set has to be constructed, R_{sim} as a 2-tuple such that $R_{sim} = (C, BC)$, where C is a cluster that belongs to C_{set2} and BC is a non-empty finite set of similar clusters correspondences which pertain to C_{set1}. Then, for each entry pair inside R_{sim}, three strings/sequences are generated during data preparation, an input source *(S)*, a target pattern *(P)*, and a distinct pattern, *(PR)*. They are created by applying normalization methods to C and its related BC. The input source, *S,* and the target pattern, *P*, are formed by concatenating tokens and prefacing them with a separator. The separator facilitates GPU token differentiation and matching, while a distinct pattern, *PR*, removes the repetition of the character set in the target pattern *P*.

The proposed matching algorithm, PPM, is based around two scalability techniques (Rahm, 2011) used in the specialized literature to reduce the time and space complexity involved in matching large-scale metadata models: partitioning and parallel strategies. It is responsible for dispatching pairs of the similarity set consecutively and evolving the concept of SIMD to flow through cores in order to take advantage of the robust GPU architectures to improve the performance of a Levenstein distance matcher. The F-measure is used to assess the alignment quality, defined as follows:

$$\mathrm{Re}c = \frac{|R \cap A|}{R}, \mathrm{Pr}ec = \frac{|R \cap A|}{A}, \text{ and}$$

$$F - m = \frac{\mathrm{Re}c \times \mathrm{Pr}ec}{\mathrm{Re}c + \mathrm{Pr}ec}$$

where R is the alignment produced from the domain and A is the alignment generated by the matching process. Metrics of *Rec* and *Prec* show the completeness and accuracy of the alignments, which are often balanced by their reconciliation mean, i.e., *F-m*.

4. PPM Matching Algorithm

The transition from traditional sequential algorithms for ASM with k-differences to their dynamic programming model allows for widespread usage in numerous applications (Navarro, 2001). Nonetheless, it suffers from data dependency, making it challenging to design a parallel algorithm for approximate string matching with k-differences (Guo et al., 2013). This risk has to be discarded when matching the large-scale scene of metadata models. Hence, this section introduces PPM, a parallel matching approach for large-scale metadata models with several benefits. It is a pairwise matching approach for metadata models, which delivers sequential pairings of the similarity set to the GPU. Thus, it reduces the amount of data transmitted through the GPU. Then, using the proposed parallel methodology of dynamic approximate string matching, each cluster string *P* is matched to its associated bulk of similar clusters string *S*. Furthermore, PPM removes data dependency impediments in the matching space. The algorithm can be implemented by constructing a set of similarity matrices with varying parallel matching performance, as shown in Figure 3. The edit distance component of the parallel pairwise clustering matching approach, PPM, significantly relies on the work described in (Guo et al., 2013). However, this effort must

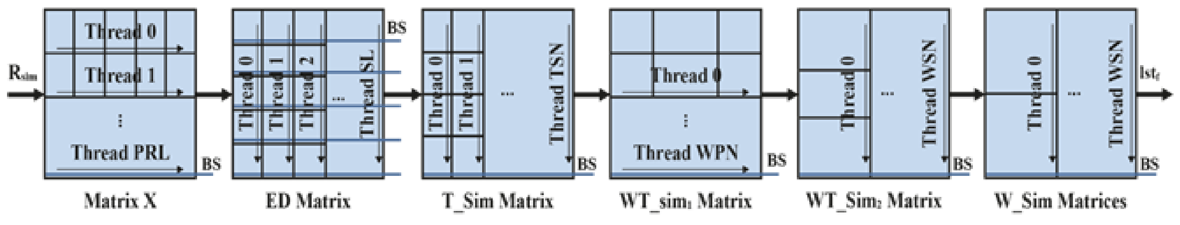

Figure 3. PPM matching algorithm

be modified to accommodate the structured data of the given metadata models so relevant entities can be easily distinguished on the GPU.

The edit distance calculation consists of two primary steps: a parallel building of matrix *X* and a parallel filling of the edit distance matrix *ED*. After establishing *ED* matrix, getting relevant entities across concatenated strings within the GPU is necessary. This stage develops parallel construction for a different set of similarity matrices, including the tokens similarity matrix *T_Sim*, words-tokens similarity matrices, WT_Sim_1, WT_Sim_2, and words similarity matrices, *W_Sim Matrices*.

4.1 Approximate String Matching

Let Σ be the alphabet and $\Sigma*$ is the set of strings over Σ. $\lambda \notin \Sigma$ is the null string. A string $x \in \Sigma*$ is denoted as $x = x_1x_2x_3...x_n$, where x_i is the i^{th} symbol of $x, x_i...x_j$, is referred as the substring of x including the symbols from x_i to x_j, $1 \leq i \leq j \leq n$, its length is defined as $| x_i...x_j | = j - i + 1$ and it is the null string $\lambda(|\lambda| = 0)$ if $i > j$. An edit operation is a pair $(a, b) \neq (\lambda, \lambda)$ of strings less than or equal to 1 and usually written as $a \to b$. a → b is called a change operation if $a \neq \lambda$ and $b \neq \lambda$, a delete operation if $b = \lambda$; and an insert operation if $a = \lambda$. An edit transformation of two strings *x* and y is a sequence of elementary edit operations which converts *x* into *y* and it is denoted as $T_{x,y} = T_1T_2...T_l$. Let *v* be an arbitrary cost function which assigns to each edit operation $a \to$ b a non-negative real number $v\ (a \to b)$, and the weight of an edit transformation $T_{x,y}$ can be computed by $v(\mathrm{T}_{x,y}) = \sum_{i}^{l} y(T_i)$. Then, let the edit distance $\varrho\ (x, y)$ from string *x* to string *y* be the minimum cost of all sequences of edit operations which transform *x* to *y*. The approximate string matching (ASM) problem is to find the edit distance between two strings.

4.2 Dynamic Programming Problem

Given two tokens $tok_P \in P$, $tok_S \in S$, $tok_P = [0, ..., p-1]$, $tok_S = [0, ..., q-1]$, and a non-negative integer *k*, $0 \leq k \leq q$, the minimum edit distance between the two tokens is to find out all locations l and $t-1$ in tok_S where $0 \leq l \leq t-1 \leq q-1$, such that the edit distance $ED\ (tok_P = [0, ..., p-1], tok_S = [l, ..., t-1]) \leq k$. The edit distance formula $ED(u, v)$ can be represented by the recurrence in equation 1. $ED(u, v)$ is the distance between the first *u* characters of tok_P and the first *v* characters of tok_S. Dynamic programming of traditional approximate string matching with k-differences is not optimal due to its severe limitations. Initially, the sequential calculation of the table requires *O (pq)* time and *q+3p+4* sequential complexity (Guo et al., 2013). As demonstrated by equation 1, each element of the edit distance matrix is dependent on its predecessors in the same row or column $ED(u-1, v-1)$, $ED(u-1, v)$, and $ED(u, v-1)$. In the present work, the prior neighboring element on the same row poses the most significant barrier to developing an ASM algorithm with k-differences based on row parallelization suitable for large-scale metadata models.

4.3 Edit Distance Calculation

It is known that the computation of *ED [u, v*] in the sequential technique depends on the value that resides in the same row or column; therefore, this is not adaptable to parallel processing.

$$ED(u,v) = \begin{cases} v, & if\ u = 0 \\ u, & if\ v = 0 \\ ED(u-1, v-1), & if\ tok_p(u-1) = tok_s(v-1) \\ 1 + \min(ED(u, v-1), ED(u-1, v-1), ED(u-1, v)), & o.w. \end{cases} \quad (1)$$

GPU solutions have been developed to accelerate the ASM task, which may be categorized into two primary forms of parallel processing: diagonal flow computing (Chen et al., 2017) and row flow computing (Guo et al., 2013). The computation of the diagonal flow is based on the parallel processing of the edit distance matrix for each element in the same diagonal flow. The length of the input pattern limits the maximum number of threads processed concurrently. This parallel scheme suffers from insufficient parallel expandability, especially when adopting GPUs.

On the other hand, row flow computing parallelizes the processing of the edit distance matrix elements in the same row. Row flow computing increases parallelism and decreases synchronization compared to the diagonal parallel technique. The proposed PPM offers a method for removing the data dependency problem based on row flow computing.

A matrix *X* has to be constructed whose dimension is $|\Sigma| \times SL$, where $|\Sigma|$ is the size of the character set in the PR string, abbreviated as *PRL*, and *SL* is the length of the input string *S*. *PR*[$0,...,|\Sigma| - 1$] refers to characters in Σ. The matrix *X* [*u*, *v*] can be calculated using equation (2), where *u* and *v* are the indices devoted for each matching inside GPU. Each character in the character set of the input string *S* is indexed, assuming that each token is viewed as an independent matching unit. At the start of each token, the index becomes 0. This set is known as *SIndices*.

$$ED(u,v)=\begin{cases}0, & if\ S[v]=',' \\ SIndices[v], & if\ S[v]=PR[u] \\ X[u,v-1], & ow.\end{cases} \quad (2)$$

To compute matrix *X*, because the data of each row are independent, data in each row can be computed in parallel. Additionally, the barrier sync, BS, is called to verify completing all rows before calculating the edit distance matrix.

According to matrix *X*, it is possible to redraft equation (1) such that computation of the data of the *u-th* row of the edit distance matrix, *ED* depends exclusively on the data of the *(u-1)-th* row. *minVal* represents the minimum value between the neighboring elements in the previous row, $ED(u-1,v)$, $ED(u-1,v-1)$. This can be explained using equation (3), where *PIndices* is a set of indices for characters inside the target pattern, *P*, regenerated for each token. *k* is the location of *P[u]* inside matrix *X* where $1 \le u \le PL$ and *PL* is the total length of the target pattern. All elements in each row can be processed in parallel. In contrast to matrix *X* computation, each row has to be finished before proceeding to the next. Barrier sync, *BS*, ensures that all threads can process all elements of a row before proceeding to the next row. Algorithm 1 is used to implement both steps of the proposed PPM matching strategy: construction of the matrix *X* and computation of the *ED* matrix.

4.4 Computation of Entities Similarities

This stage aims to retrieve corresponding entities across concatenated strings inside the GPU. This can be realized across the parallel construction for a set of similarity matrices; tokens similarity matrix *T_Sim*, words-tokens similarity matrices, *WT_Sim*$_1$ *Matrix, WT_Sim*$_2$ *Matrix,* and words similarity matrices *W_Sim Matrices*.

Tokens similarity matrix, *T_Sim*, calculation has to be performed after finalizing *ED* matrix construction. Through *ED* matrix, the bottom-right elements of the edit distance spaces of the pairs of tokens are collected. Due to the non-dependency between rows, threads work on the same row to compute all elements in parallel without synchronization. However, the barrier sync must be applied after the entire matrix building for the next step. *T_Sim(u, v)* can be determined through equation (4). It is based upon two parameters, *var1*, and *var2*, given in equations (5) and (6), respectively. Token counts are stored in *TSN* and *TPN*. *SSIndices* and *PSIndices* represent the occurrence of a separator along the input string *S*, and the target pattern *P*, respectively. *STL* and *PTL* are abbreviations for the lists of all token lengths of the input string *S*, and the target pattern *P*, respectively.

$$ED(u,v)=\begin{cases}SIndics[v], & if\ PIndices[u]=0 \\ PIndices[u], & if\ SIndices[v]=0 \\ ED(u-1,v-1), & if\ S[v]=P[u] \\ 1+\min(\min Val, PIndices[u]+SIndices[v]-1), & if\ X[k,v]=0 \\ 1+\min(\min Val, ED[u-1,v-SIndices[v]+X[k,v]-1)+SIndices[v]-1-X[k,v], & ow.\end{cases} \quad (3)$$

Algorithm 1. GPU kernel pseudo-code of Matrix *X* construction and *ED* Matrix computation of PPM Algorithm

```
1.  Input: SL, S, SIndices, PL, P, PIndices, PRL, PR.
2.  Output: X, ED.
3.  FOR ALL $u \in PRL$ PARALLEL DO  //Matrix X
4.    FOR $v \leftarrow 0$ to SL DO
5.      Compute X[u, v] using equation 2;
6.    END FOR
7.  END PARALLEL FOR
8.  Barrier synchronization;
9.  FOR $u \leftarrow 0$ to PL DO            //ED Matrix
10.   FOR ALL $v \in SL$ PARALLEL DO
11.     Compute ED(u, v) using equation 3;
12.   END PARALLEL FOR
13.   Barrier synchronization;
14. END FOR
```

$$T\_Sim[u,v]=1-\frac{ED(\text{var}1,\text{var}2)}{\max(STL[v],PTL[u])} \quad (4)$$

where *var1* and *var2* can be determined as follows:

$$\text{var}1=\begin{cases}PL-1, & if\ u=TPN-1 \\ PSIndices[u+1]-1, & ow.\end{cases} \quad (5)$$

$$\text{var}2=\begin{cases}SL-1, & if\ v=TSN-1 \\ SSIndices[v+1]-1, & ow.\end{cases} \quad (6)$$

Words-tokens similarity matrices are of two kinds: the first words-tokens similarity matrix, WT_Sim_1, and the second words-tokens similarity matrix, WT_Sim_2. They have different parallelization natures subject to their subordinate to the input string S or the target pattern P. In WT_Sim_1, a single thread works on the same row inside the matrix. Each S token must be compared to all tokens of each word of P in their matching area. Hence, the entries of the matrix are filled with the largest similarity values. On the contrary, WT_Sim_2 has all threads working on the same row within the matching area without synchronization. Each P token is compared to all tokens of each word of S in their matching area, and then the greatest similarity values are computed. Finally, the barrier sync *BS* must be used after each matrix construction to complete the next step. Tokens similarity and words-tokens similarity matrices are implemented using Algorithm 2. *SWTSFIndices* and *PWTSFIndices* hold the start and end indices for each word token in the input string and the target pattern, respectively.

Three-word similarity matrices are filled in the same parallelization manner. Parallelization is performed in all matching regions on a single row. The first matrix W_Sim_1 is based on the first words-tokens similarity matrix WT_Sim_1, in which each thread sums the maximum similarities of a single word in P against its tokens in S in their matching area. The second matrix W_Sim_2 is based on the second words-tokens similarity matrix WT_Sim_2. Each thread sums the maximum similarities of a single word in S against its tokens in P in their matching area. The third matrix W_Sim stores the final output transferred from the device to the host. A single thread sums the values in the prior two matrices for a pair of words divided by the sum of counts of their tokens. Thus, the final output holds the similarity values between words of the input string against words of the target pattern and can be estimated using equation 7. Algorithm 3 is used to implement words similarity matrices. The counts of words of the input string and the target pattern are set into *WSN* and *WPN*, respectively.

$$sumFinal = \frac{(W\_Sim_1[u,v] + W\_Sim_2[u,v])}{((SWTSFIndices[v+1] - SWTSFIndices[v]) + (PWTSFIndices[u+1] - PWTSFIndices[u]))} \quad (7)$$

Algorithm 2. General GPU kernel pseudo-code of Tokens Similarity and Words-Tokens Similarity Matrices

```
1.  Input: SL, TSN, SWTSFIndices, SSIndices, WSN PL, TPN, PWTSFIndices, PSIndices, WPN.
2.  Output: T_Sim, WT_Sim_1, WT_Sim_2.
3.  FOR u ← 0 to TPN DO                                   //T_sim
4.     FOR All v ∈ TSN PARALLEL DO
5.        Compute T_sim[u, v] using equation 4;
6.     END PARALLEL FOR
7.  END FOR
8.  Barrier synchronization;
9.  FOR ALL u ∈ WPN PARALLEL DO                           //WT_Sim_1
10.    FOR v ← 0 to TSN DO
11.       max = 0.0f;
12.       FOR k ← PWTSFIndices [u] to PWTSFIndices [ u + 1] DO
13.          Result = T_sim[k , v];
14.          IF Result > max THEN
15.             max = Result;
16.       END FOR
17.       WT_Sim_1[u, v] = max;
18.    END FOR
19. END PARALLEL FOR
20. Barrier synchronization;
21. FOR u ← 0 to TPN DO                                   //WT_Sim_2
22.    FOR ALL v ∈ WSN PARALLEL DO
23.       max = 0.0f;
24.       FOR k ← SWTSFIndices[v] to SWTSFIndices [v + 1] DO
25.          Result = T_sim[ u , k];
26.          IF Result > max THEN
27.             max = Result;
28.       END FOR
29.       WT_Sim_2[u, v] = max;
30.    END PARALLEL FOR
31. END FOR
32. Barrier synchronization;
```

Algorithm 3. GPU kernel pseudo-code of Words Similarity Matrices

```
1.  Input: WPN, SWTSFIndices, WSN, PWTSFIndices.
2.  Output: W_Sim_1, W_Sim_2, W_Sim.
3.  FOR v ← 0 to WSN DO                                   //W_Sim_1
4.     FOR ALL u ∈ WPN PARALLEL DO
5.        Sum = 0.0f;
6.        FOR k ← SWTSFIndices[v] to SWTSFIndices[v + 1] DO
7.           Sum+ = WT_Sim_1[u , k];
8.        END FOR
9.        W_Sim_1[u, v] = Sum;
10.    END PARALLEL FOR
11. END FOR
12. Barrier synchronization;
13. FOR u ← 0 to WPN DO                                   //W_Sim_2
14.    FOR ALL v ∈ WSN PARALLEL DO
15.       Sum = 0.0f;
16.       FOR k ←PWTSFIndices[u] to PWTSFIndices [u+1] DO
17.          Sum+ = WT_Sim_2[k, v];
18.       END FOR
19.       W_Sim_2[u, v] = Sum;
20.    END PARALLEL FOR
21. END FOR
22. Barrier synchronization;
23. FOR u ← 0 to WPN DO                                   //W_Sim
24.    FOR ALL v ∈ WSN PARALLEL DO
25.       Calculate sumFinal using equation 7
26.       W_Sim [u, v] = sumFinal;
27.    END PARALLEL FOR
28. END FOR
29. Barrier synchronization;
```

As a supplemental contribution, all independent matching tasks can be accumulated and sent to the device into two batches, one for the input string and another for the target pattern. Then individual matching tasks can be isolated and executed in parallel, one after the other. It is called the Parallel Accumulated Matching strategy for metadata models, PAM. PPM and PAM have the same implementation except for displacements that enable accessibility through clusters and structures inside GPU in the PAM alternative.

5. Experimental Evaluation

In this section, PPM matching algorithm is tested on various real-world schemas and ontologies, in order to determine if the matching efficiency could be improved, while preserving quality regarding precision, recall, and F-measure. In all experiments, PPM has been compared against dynamic approximate string matching, ASM, and parallel accumulated matching strategy, PAM. The experiments have been carried out on Intel Core i7-4710HQ, 2.5GHz, and installed memory (RAM) of 16 GB. It has dual display adapters, Intel(R) HD Graphics 4600, and NVIDIA GeForce GTX 860M. The algorithm has been implemented using JCuda 10.0.0, CUDA 10.0 on the Java environment of jdk-15.0.1, and Apache-NetBeans 12.0 for Windows 10.

5.1 Schemas Test

Schemas describe the structure and legal building blocks of XML documents. In the proposed evaluation, a dataset that consists of a number of real-world mapping tasks from various domains is exploited (Institute for Informatics. Georg-August-Universität Göttingen, n. d.; UW CSE Department Data Set, n. d.), ranging in size from small to large.

5.1.1 Efficiency Evaluation

In this track, the performance of the proposed parallel matching strategy PPM is compared to those of the PAM and ASM strategies. A threshold of 0.2 is used to detect the most comparable clusters, and a threshold of 0.5 is utilized for matching. Each matching task was executed five times with identical parameters, and the average value for each test was calculated. According to the number of matching candidates, the efficiency results are listed in descending order, as shown in Figure 4.

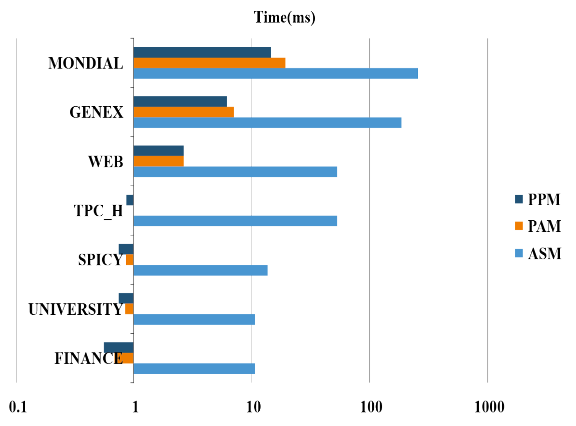

Figure 4. Efficiency on schemas

The graph shows that execution times increase as cluster size increases, but decrease as GPU performance improves. The graph shows that MONDIAL achieves the best timing performance, while FINANCE achieves the worst, across all the tested algorithms. For MONDIAL, the performance of PAM and PPM on GPU was achieved 13.48 and 17.77 times faster than that of ASM. For GENEX, PAM and PPM attained the performance 26.89 and 30.70 times faster than ASM. For WEB, the speedups of PAM and PPM achieved 19.97 and 20.25 times faster than those of ASM. For TPC_H, PAM and PPM achieved speedups 19.6 and 22.47 times faster than those of ASM. For SPICY, parallel algorithms PAM and PPM are estimated to be 16.03 and 18.57 times faster than those of ASM. For the popular UNIVERSITY schema, PAM and PPM outperformed those of ASM by 12.89 and 14.84 times. Finally, for FINANCE, PAM and PPM perform 16.29 and 19.48 times faster than those of ASM. In this regard, the PPM parallel matching strategy outperforms the PAM parallel matching algorithm on GPU and the ASM serial algorithm on CPU.

5.1.2 Effectiveness Evaluation

Through the experiment, the consistency of the matching quality was evaluated across the compared implementations utilized in the efficiency evaluation test, at varying threshold numbers (0.1-0.9). It is deduced that all parallel matching techniques have maintained the same matching quality at any selected threshold. As it can be seen in Figure 5, the threshold that produces the maximum quality of matching was selected for each schema.

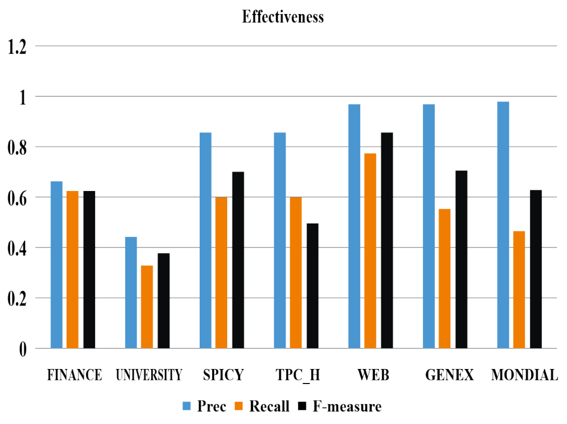

Figure 5. Matching Quality Comparisons

5.2 Ontology Test

Additional tests were conducted to validate the performance of the proposed parallel PPM algorithm using real-world ontologies from the OAEI dataset (Ontology Alignment Evaluation Initiative, n. d.). To determine which matcher is superior for accelerating GPU performance, all matchers are evaluated using benchmarks and anatomy matching tracks. These tracks vary in size and cover various facets of the ontology matching problem.

5.2.1 Benchmarks Track

The benchmark test set is based on a seed ontology and its many variants. Variations are created by removing and modifying features from the seed ontology. Entity names, comments, specialization hierarchy, instances, properties, and classes are all considered features.

Table 1. Efficiency Evaluation of Benchmarks at threshold 0.2 for VSM, 0.5 for matching (ms)

Test	Prec.	Rec.	ASM	PAM	PPM	Test	Prec.	Rec.	ASM	PAM	PPM	Test	Prec.	Rec.	ASM	PAM	PPM
101	0.5	1.0	51.26	7.23	6.12	248-6	0.48	0.22	33.56	4.26	3.55	257-2	0.44	0.76	40.88	3.99	3.16
201	0.41	0.27	37.03	5.02	4.10	248-8	0.48	0.21	42.62	4.72	3.99	257-4	0.48	0.58	31.71	3.56	2.83
201-2	0.48	0.84	50.73	7.36	5.93	249	0.33	0.01	13.53	1.81	1.54	257-6	0.47	0.42	27.00	3.33	2.64
201-4	0.47	0.72	50.93	7.11	5.44	249-2	0.49	0.77	51.73	6.83	5.60	257-8	0.50	0.21	20.00	2.39	1.91
201-6	0.45	0.60	48.09	6.69	5.52	249-4	0.48	0.58	53.91	6.48	5.34	258	0.33	0.01	13.24	1.38	1.17
201-8	0.44	0.43	39.89	6.00	5.01	249-6	0.47	0.40	48.38	6.12	5.04	258-2	0.51	0.77	56.70	6.89	5.60
202	0.33	0.01	13.63	1.84	1.66	249-8	0.48	0.21	35.68	4.51	3.73	258-4	0.51	0.58	50.78	6.52	5.29
202-2	0.49	0.77	45.30	6.73	5.45	250-2	0.44	0.76	36.85	3.82	3.02	258-6	0.51	0.39	45.05	6.09	5.93
202-4	0.49	0.58	56.65	6.53	5.32	250-4	0.46	0.58	34.30	3.70	2.96	258-8	0.49	0.20	36.25	4.94	4.08
202-6	0.46	0.4	50.30	5.99	5.08	250-6	0.45	0.42	23.97	3.22	2.57	259	0.33	0.01	14.60	1.90	1.60
202-8	0.48	0.21	38.91	4.77	3.95	250-8	0.5	0.21	19.34	2.37	1.97	259-2	0.45	0.77	66.49	7.40	5.93
221	0.51	1.0	60.55	8.0	6.40	251	0.33	0.01	12.04	1.79	1.54	259-4	0.44	0.76	59.50	7.06	5.80
222	0.52	1.0	56.21	7.67	6.15	251-2	0.51	0.77	53.74	6.92	5.65	259-6	0.44	0.77	61.76	7.26	5.94
223	0.41	0.99	76.4	8.67	6.90	251-4	0.5	0.58	49.04	6.35	5.13	259-8	0.45	0.77	55.08	6.99	5.76
224	0.5	1.0	51.18	7.64	6.34	251-6	0.51	0.4	41.19	5.97	4.84	260	0.0	0.0	13.08	1.38	1.14
225	0.5	1.0	62.11	8.00	6.59	251-8	0.48	0.22	33.56	4.26	3.55	260-2	0.42	0.76	35.41	3.56	2.84
228	0.41	1.0	41.52	4.37	3.44	252	0.33	0.01	13.65	1.84	1.62	260-4	0.47	0.59	30.46	3.47	2.82
232	0.5	1.0	60.28	7.98	6.45	252-2	0.4	0.77	58.16	7.26	5.84	260-6	0.48	0.41	26.04	3.06	2.40
233	0.44	1.0	37.13	4.71	3.67	252-4	0.4	0.77	62.76	7.19	5.83	260-8	0.41	0.24	19.66	2.38	1.91
236	0.42	1.0	50.43	4.31	3.37	252-6	0.4	0.77	62.07	7.20	5.80	261	0.0	0.0	11.85	1.25	1.03
237	0.52	1.0	59.44	7.64	6.17	252-8	0.4	0.77	59.37	7.33	5.86	261-2	0.33	0.76	44.50	5.42	4.34
238	0.4	0.99	84.70	8.95	7.11	253	0.33	0.01	10.25	1.48	1.22	261-4	0.32	0.73	43.50	5.48	4.38
239	0.4	1.0	47.65	4.21	3.13	253-2	0.47	0.77	61.51	7.30	5.9	261-6	0.31	0.73	47.36	5.55	4.48
240	0.26	0.97	78.31	7.7	6.19	253-4	0.46	0.58	54.14	6.70	5.44	261-8	0.34	0.76	42.81	5.47	4.31
241	0.43	1.0	42.46	4.84	3.71	253-6	0.45	0.40	46.42	6.15	5.08	262-2	0.46	0.76	31.94	4.00	3.05
246	0.4	1.0	45.55	4.26	3.33	253-8	0.47	0.21	38.22	4.94	3.96	262-4	0.49	0.58	28.74	3.57	2.76
247	0.26	0.97	77.52	7.45	7.45	254-2	0.46	0.76	29.88	3.90	3.06	262-6	0.45	0.42	23.16	3.10	2.42
248	0.33	0.01	12.19	1.8	1.62	254-4	0.44	0.58	32.85	3.43	2.68	262-8	0.5	0.21	19.22	2.09	1.73
248-2	0.48	0.78	58.29	6.86	5.69	254-6	0.45	0.42	24.46	3.02	2.43	265	0.0	0.0	14.08	1.44	1.19
248-4	0.47	0.58	51.95	6.47	5.18	254-8	0.5	0.21	17.54	2.15	1.81	266	0.0	0.0	13.32	1.20	0.98

To perform the test, the bibliographic seed ontology is chosen. Since the beginning of OAEI campaigns, it has served as the primary reference ontology. The data set contains 94 ontology pairs. First, the performance of the proposed parallel PPM matching strategy is evaluated through a series of experiments using ontologies from the OAEI benchmarks track. A 0.2 threshold determines the most similar clusters, and a 0.5 threshold is used for matching. Each matching task was done with the same settings five times, and the average value was calculated for each test. Table 1 shows the comparisons of the present tests. It conveys the precision, recall, and F-measure values for each pair in the benchmarks track. PPM and PAM have demonstrated the same matching quality as their serial counterpart ASM. Table 1 also compares the efficiency of PPM and PAM to that of ASM. Compared to ASM, PAM achieves speedups ranging from 6.64 to 11.70 times, and the performance of PPM was improved, ranging from 7.79 to 14.95 times.

5.2.2 Anatomy Track

The primary goal of the anatomy matching track is to find an alignment between the Adult Mouse Anatomy (2744 concepts) and a part of the NCI Thesaurus (3304 concepts) describing human anatomy. Due to the limited resources of the GPU, it was not feasible to match the anatomy data set using PPM and PAM parallel matching strategies. Therefore, for the whole anatomy dataset, the final similarity structure of the preparation module, R_{sim} is divided into 12 divisions; each is treated as a standalone dataset and is emitted to GPU individually.

A 0.6 threshold is used to discover the most comparable clusters and a 0.5 threshold is used to match them. Each matching task was run five times with the same settings, and the average time for each test was calculated in milliseconds. The quality and efficiency evaluation tests are collected in Table 2. For PAM, the achievements in timing performance due to divisions attain speedups ranging from 4.76 to 7.52 times faster than dynamic ASM, while PPM enhances the speed from 5.84 to 11.85 times compared to dynamic ASM. The achievement in timing performance of the whole anatomy track acquires 5.21 times more rapidly than dynamic ASM for the proposed parallel PPM matching algorithm. Still, for PAM, it reaches 4.26 times faster than dynamic ASM.

Table 2. Efficiency Evaluation of Anatomy at threshold 0.6 for VSM, 0.5 for matching (ms)

Part. No	Prec.	Rec.	ASM	PAM	PPM
1	0.22	0.02	35.72	6.47	5.23
2	0.15	0.01	30.96	6.24	5.09
3	0.13	0.02	34.20	7.18	5.85
4	0.18	0.04	102.98	13.69	11.63
5	0.15	0.06	250.65	25.53	21.16
6	0.2	0.05	129.07	18.00	14.91
7	0.18	0.03	114.88	14.51	12.10
8	0.24	0.04	93.09	13.48	11.24
9	0.23	0.04	89.03	13.39	10.93
10	0.22	0.04	104.00	14.19	11.89
11	0.16	0.03	123.09	16.21	13.32
12	0.18	0.03	86.75	14.74	12.26
Total	0.18	0.40	699.24	164.00	134.31

6. Conclusion

Metadata models are becoming increasingly popular to share and reuse knowledge. This has resulted in the developing of large-scale independent metadata models within the same or separate domains, with some information overlapping. Automatic matching has become a mandatory solution. However, the process of matching large metadata models is time and space-consuming. To this end, the work proposed in this paper presents PPM (Parallel Pairwise Matching), an efficient hardware implementable matching algorithm on GPU for large-scale metadata models when clustering techniques exist. It is based on approximate string-matching using k-difference (ASM). It dispatches pairs of the similarity set to GPU in a consecutive manner and relies upon row flow computing that hinders data dependency obstacles. It is proven that PPM is a promising approach to accelerate large-scale metadata model matching. It overcomes ASM and the parallel accumulated matching algorithm, PAM that simultaneously migrates all structures and clusters to the device. To maintain stability, a method for dividing long bulks should be included so that entities cannot be intersected. This improves matching efficiency while maintaining matching quality without requiring it to be promoted to a higher resource level.

REFERENCES

Abadi, M. & Zamanifar, K. (2011) Producing complete modules in ontology partitioning. In: *2011 International Conference on Semantic Technology and Information Retrieval*, *27 – 29 June 2011, Putrajaya, Kuala Lumpur, Malaysia*. New Jersey, SUA, Institute of Electrical and Electronics Engineers (IEEE). pp. 137-143.

Algergawy, A., Massmann, S. & Rahm, E. A. (2011) A Clustering-Based Approach for Large-Scale Ontology Matching. In: Eder, J., Bielikova, M. & Tjoa, A. M. (eds.) *Advances in Databases and Information Systems: Proceedings of the 15th International Conference (ADBIS 2011)*, *20 – 23 September 2011, Vienna, Austria*. Berlin, Heidelberg, Springer. pp. 415-428. doi: 10.1007/978-3-642-23737-9_30.

Algergawy, A., Moawed, S., Sarhan, A., Eldosouky, A. & Saake, G. (2014) Improving clustering-based schema matching using latent semantic indexing. In: Hameurlain, A., Küng, J. Wagner, R., Catania, B., Guerrini, G., Palpanas, T., Pokorný, J. & Vakali, A. (eds.) *Transactions on Large-Scale Data and Knowledge-Centered Systems XV: Selected Papers from the 17th East-European Conference on Advances in Databases and Information Systems (ADBIS 2013) Satellite Events, 1 – 4 September 2013, Genova, Italy*. Berlin, Heidelberg, Springer. pp. 102-123. doi: 10.1007/978-3-662-45761-0.

Amin, M., Batool, R., Khan, W., Lee, S. & Huh, E. (2014) SPHeRe: A Performance Initiative Towards Ontology Matching by Implementing Parallelism over Cloud Platform. *The Journal of Supercomputing*. 68(1), 274-301. doi: 10.1007/s11227-013-1037-1.

Amin, M., Khan, W., Hussain, S., Bui, D., Banos, O., Kang, B. & Lee, S. (2016) Evaluating large-scale biomedical ontology matching over parallel platforms. *Institution of Electronics and Telecommunication Engineers* (*IETE) Technical Review*. 33(4), 415-427. doi: 10.1080/02564602.2015.1117399.

Apache Jena. (n. d.) https://jena.apache.org/ [Accessed 1st August 2022].

Babalou, S., Kargar, M. & Davarpanah, S. (2016) Large-scale ontology matching: a review of the literature. In: *2016 Second International Conference on Web Research (ICWR), 27 – 28 April 2016, Tehran, Iran.* New York, USA, Institute of Electrical and Electronics Engineers (IEEE). pp. 158-165. doi: 10.1109/ICWR.2016.7498461.

Chen, X., Wang, C., Tang, S., Yu, C. & Zou, Q. (2017) CMSA: a heterogeneous CPU/GPU computing system for multiple similar RNA/DNA sequence alignment. *BMC Bioinformatics*. 18, 1-10. doi: 10.1186/s12859-017-1725-6.

Chong, I. & Lee, S. (2022) Deep learning based semantic ontology alignment process and predictive analysis of depressive disorder. In: *2022 International Conference on Information Networking (ICOIN)*, *12 – 15 January 2022, Jeju-si, Republic of Korea*. New York, USA, Institute of Electrical and Electronics Engineers (IEEE). pp. 164-167. doi: 10.1109/ICOIN53446.2022.9687251.

Coyle, K. (2010) Metadata models of the world wide web. *Library Technology Reports*. 46(2), 12-19.

Cuenca Grau, B., Horrocks, I., Kazakov, Y. & Sattler, U. (2007) A Logical Framework for Modularity of Ontologies. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), 6 – 12 January 2007, Hyderabad, India*. Menlo Park, USA, AAAI Press. pp. 298–303.

Cuenca Grau, B., Horrocks, I., Kazakov, Y. & Sattler, U. (2008) Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research*. 31, 273-318. doi: 10.1613/jair.2375.

Djeddi, W., Khadir, M. & Ben-Yahia, S. (2014) XMap++: results for OAEI 2014. In: *Proceedings of the 9th International Workshop on Ontology Matching*, *20 October 2014, Riva del Garda, Trentino, Italy*. pp. 163-169.

Do, H. & Rahm, E. (2007) Matching large schemas: Approaches and evaluation. *Information Systems*. 32(6), 857-885. doi: 10.1016/j.is.2006.09.002.

El Abdi, M., Souid, H., Kachroudi, M. & Yahia, S. (2015) CLONA results for OAEI 2015. In: *Proceedings of the 10th International Workshop on Ontology Matching, 12 October 2015, Bethlehem, PA, USA*. pp. 124-129.

Groß, A., Hartung, M., Kirsten, T. & Rahm, E. (2010) On matching large life science ontologies in parallel. In: Lambrix, P. & Kemp, G. (eds.) *Data Integration in the Life Sciences: Proceedings of the 7th International Conference DILS 2010*, *August*, *Gothenburg, Sweden.* Berlin, Heidelberg, Springer. pp. 35-49. doi: 10.1007/978-3-642-15120-0_4.

Groß, A., Hartung, M., Kirsten, T. & Rahm, E. (2012) GOMMA results for OAEI 2012. In: *Proceedings of the 7th International Workshop on Ontology Matching*, *11 November 2012, MA, USA.* pp. 133–140.

Guo, L., Du, S., Ren, M., Liu, Y., Li, J., He, J., Tian, N. & Li, K. (2013) Parallel Algorithm for Approximate String Matching with K Differences. In: *2013 IEEE Eighth International Conference on Networking, Architecture and Storage, 17 – 19 July 2013, Xi'an, China*. New York, USA, Institute of Electrical and Electronics Engineers (IEEE). pp. 257-261. doi: 10.1109/NAS.2013.40.

Huber, J., Sztyler, T., Noessner, J. & Meilicke, C. (2011) CODI: Combinatorial optimization for data integration – results for OAEI 2011. In: *Proceedings of the 6th International Conference on Ontology Matching, 24 October 2011, Bonn, Germany.* pp. 142-146.

Institute for Informatics. Georg-August-Universität Göttingen. (n. d.) *The MONDIAL Database*. http://www.dbis.informatik.uni-goettingen.de/Mondial/ [Accessed 5th April 2022].

Java Enterprise Edition (Java EE). (n. d.) https://xsom.java.net/ [Accessed 1st January 2022].

Jung, H., Yoo, S. & Park, S. (2012) Context Modelling Using Semantic Web Technologies. *Studies in Informatics and Control*. 21(2), 173-180. doi: 10.24846/v21i2y201207.

Kirsten, T., Groß, A., Hartung, M. & Rahm, E. (2011) GOMMA: a component-based infrastructure for managing and analyzing life science ontologies and their evolution. *Journal of Biomedical Semantics*. 2(1): 6. doi: 10.1186/2041-1480-2-6.

Kusnierczyk, W. (2008) Taxonomy-based partitioning of the Gene Ontology. *Journal of Biomedical Informatics*. 41(2), 282-292. doi: 10.1016/j.jbi.2007.07.007.

Lee, M. L., Yang. L. H., Hsu. W. & Yang, X. (2002) XClust: clustering XML schemas for effective integration. In: *Proceedings of the International Conference on Information and Knowledge Management (CIKM '02), 4 – 9 November 2002, McLean, Virginia, USA*. New York, USA, Association for Computing Machinery (ACM). pp. 292–299. doi: 10.1145/584792.584841.

Mani, S. & Annadurai, S. (2021) Explicit Link Discovery Scheme Optimized with Ontology Mapping using Improved Machine Learning Approach. *Studies in Informatics and Control*. 30(1), 67-75. doi: 10.24846/v30i1y202106.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*. 33(1), 31-88. doi: 10.1145/375360.375365.

Ngo, D. & Bellahsene, Z. (2016) Overview of YAM++—(not) Yet Another Matcher for ontology alignment task. *Journal of Web Semantics*. 41, 30-49. doi: 10.1016/j.websem.2016.09.002.

Ochieng, P. & Kyanda, S. (2018) Large-scale ontology matching: State-of-the-art analysis. *ACM Computing Surveys*. 51(4), 1-35. doi: 10.1145/3211871.

Ontology Alignment Evaluation Initiative. (n. d.) http://oaei.ontologymatching.org/ [Accessed 1st December 2022].

Otero-Cerdeira, L., Rodríguez-Martínez, F. J. & Gómez-Rodríguez, A. (2015) Ontology matching: A literature review. *Expert Systems with Applications*. 42(2), 949-971. doi: 10.1016/j.eswa.2014.08.032.

Owens, J., Houston, M., Luebke, D., Green, S., Stone, J. & Phillips, J. (2008) GPU Computing. In: *Proceedings of the IEEE*. 96(5). pp. 879-899. doi: 10.1109/JPROC.2008.917757.

Rahm, E. (2011) Towards large-scale schema and ontology matching. In: Bellahsene, Z., Bonifati, A. & Rahm, E. (eds.) *Schema Matching and Mapping*. Berlin, Heidelberg, Springer, pp. 3-27.

Saruladha, K. & Ranjini, S. (2016) COGOM: Cognitive Theory Based Ontology Matching System. *Procedia Computer Science*. 85, 301-308. doi: 10.1016/j.procs.2016.05.237.

Schadd, F. & Roos, N. (2014) Alignment evaluation of MaasMatch for the OAEI 2014 campaign. In: *Proceedings of the 9th International Workshop on Ontology Matching, 20 October 2014, Riva del Garda, Trentino, Italy*. pp. 135-141.

Schlicht, A. & Stuckenschmidt, H. (2007) Criteria-based partitioning of large ontologies. In: *Proceedings of the 4th International Conference on Knowledge Capture (K-CAP07), 28-31 October 2007, Whistler, BC, Canada*. New York, USA, Association for Computing Machinery (ACM). pp. 171-172.

Sellami, S., Benharkat, A., Amghar, Y. & Rifaieh, R. (2008) Study of Challenges and Techniques in Large Scale Matching. In: Filipe, J. & Cordeiro, J. (eds.) *Enterprise Information Systems: Proceedings of the Tenth International Conference (ICEIS), 12-16 June 2008, Barcelona, Spain*. Berlin Heidelberg, Springer. pp. 355-361. doi: 10.1007/978-3-642-00670-8.

Shvaiko, P. & Euzenat, J. (2008) Ten challenges for ontology matching. In: Meersman, R. & Tari, Z. (eds.) *On the Move to Meaningful Internet Systems: Proceedings of OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, 9 – 14, November 2008, Monterrey, Mexico*. Berlin, Heidelberg, Springer. pp. 1164-1182. doi: 10.1007/978-3-540-88873-4_18.

Shvaiko, P., Euzenat, J., Jiménez-Ruiz, E., Cheatham, M., Hassanzadeh, O. & Ichise, R. (eds.) (2016) *Proceedings of the 11th International Workshop on Ontology Matching (OM-2016), 18 October 2016, Kobe, Japan*.

Sivasankari, S. & Shomona, G. J. (2016) A Novel Semi-Automated Ontology Construction Framework (SOCF) for Psoriasis Detection: Pioneering the Psoriasis Risk Assessment Remedy (PRAR) Database. *Studies in Informatics and Control*. 25(2), 237-244. doi: 10.24846/v25i2y201611.

Tenschert, A., Assel, M., Cheptsov, A., Gallizo, G., Della Valle, E. & Celino, I. (2009) Parallelization and Distribution Techniques for Ontology Matching in Urban Computing Environments. In: *Proceedings of the 4th International Workshop on Ontology Matching, 25 October 2009, Chantilly, USA*. pp. 248-249.

Tran, T., Liu, Y. & Schmidt, B. (2016) Bit-parallel approximate pattern matching: Kepler GPU versus Xeon Phi. *Parallel Computing*. 54, 128-138. doi: 10.1016/j.parco.2015.11.001.

UW CSE Department Data Set. (n. d.) *XMLData Repository*. http://www.cs.washington.edu/research/xmldatasets/ [Accessed 5th April 2022].

Wang, P. (2010) Lily-LOM: An efficient system for matching large ontologies with non-partitioned method. In: *Proceedings of the 2010 International Conference on Posters & Demonstrations Track (CEUR Workshop), 9 November 2010, Shanghai, China*. pp. 69-72.

Xue, X., Jiang, C., Wang, H., Tsai, P., Mao, G. & Zhu, H. (2021) An improved multi-objective evolutionary optimization algorithm with inverse model for matching sensor ontologies. *Soft Computing*. 25(18), 12227-12240. doi: 10.1007/s00500-021-05895-y.

Zhang, Z., Che, H., Shi, P., Sun, Y., & Gu, J. (2006) Formulation schema matching problem for combinatorial optimization problem. *Interoperability in Business Information Systems (IBIS)*. 1(1), 33-60.