The Journal of Supercomputing manuscript No.
(will be inserted by the editor)

Exploring Hybrid Parallel Systems for Probabilistic Record Linkage

Murilo Boratto · Pedro Alonso · Clicia Pinto · Pedro Melo · Marcos Barreto · Spiros Denaxas

Received: date / Accepted: date

Abstract Record linkage is a technique widely used to gather data stored in disparate data sources that presumably pertain to the same real world entity. This integration can be done deterministically or probabilistically, depending on the existence of common key attributes among all data sources involved. The probabilistic approach is very time consuming due to the amount of records that must be compared, specifically in big data scenarios. In this paper, we propose and evaluate a methodology that simultaneously exploits multicore and multi-GPU architectures in order to perform the probabilistic linkage of large-scale Brazilian governmental databases. We present some algorithmic optimizations that provide high accuracy and improve performance by defining the best algorithm-architecture combination for a problem given its input size. We also discuss performance results obtained with different data samples, showing that a hybrid approach outperforms other configurations, providing an average speedup of 7.9 when linking up to 20.000 million records.

Keywords Probabilistic Linkage · Public Health · Performance evaluation · Multicore · Multi-GPU

1 Introduction

The task of linking multiple, disparate database records representing data from the same real world entity is known as *record linkage* [4], being a technique widely

Murilo Boratto
Núcleo de Arquitetura de Computadores e Sistemas Operacionais, Universidade do Estado da Bahia, Salvador, Bahia, Brazil, E-mail: muriloboratto@uneb.br
Pedro Alonso
Departament of Information Systems and Computation, Universitat Politècnica de València, Spain, E-mail: palonso@upv.es
Clicia Pinto · Pedro Melo · Marcos Barreto
Laboratório de Sistemas Distribuídos, Universidade Federal da Bahia, Salvador, Bahia, Brazil, E-mail: {cliciasp, pmelo, marcseb}@ufba.br
Spiros Denaxas
Institute of Health Informatics Research, School of Computer Science and Informatics, University College London, London, United Kingdom, E-mail: s.denaxas@ucl.ac.uk

used in biomedical and health research, finance, government and other domains. Specifically in Health research, data stored in disparate information systems need to be combined for diverse purposes including aggregation of medical and hospital services, assessment of public policies, track of patient care, surveillance and monitoring. This is often a challenging task as data quality, complexity and size dramatically differs among these information systems.

There are two main approaches for record linkage: deterministic and probabilistic [6]. Deterministic linkage uses one or more unique identifying (key) attributes that are common across the data sources to link records. When these common attributes are absent, a combination of meaningful attributes should be used to probabilistically link the records. In order to improve accuracy on matching decisions, probabilistic methods must perform a huge number of comparisons and calculate similarity indices. These are complex and highly time-consuming tasks.

Some techniques are used to reduce dimensionality, being *blocking* the prevalent approach. Depending on hardware capabilities, one must group the records into blocks according to some similarity criterion – usually the values of a given set of attributes – to avoid unnecessary comparisons. Although reducing the execution time, blocking can affect accuracy if records that must pertain to similar blocks are not correctly grouped.

Other important issues related to heterogeneity and privacy arise from the sensitive nature of health data. Data harmonization and anonymization must be applied prior to the record linkage step, affecting, in turn, the execution time.

Given the complexity of implementing probabilistic record linkage methods targeted to huge databases, novel algorithmic approaches able to fully exploit the processing power of multiple resources are desired. Hybrid parallel architectures can be considered a viable approach, even introducing some challenges in the design of algorithms and related system software.

Our proposal is to simultaneously use all available CPU and GPU devices present in a heterogeneous system and balance the workload among them to perform the necessary data linkage operations in a timely and accurate manner. We deal with optimal workload distributions for hybrid systems [21], efficient grid configurations, and analysis of data transfers, among other hardware specific issues.

Our research scope involves the linkage of several large Brazilian governmental databases with socioeconomic and health care data from the Brazilian Public Health System. These databases are linked in order to create accurate "data marts" (disease-specific data) used for epidemiological studies. These studies are part of three major ongoing Brazil-UK scientific collaborations: 1) *the 100 million cohort project*, targeted to assess the effects of a conditional cash transfer programme on infectious diseases from 114 million individuals; 2) a *long-term surveillance platform for Zika and microcephaly*, which aims to support a longitudinal study (2001 to 2030) of children diagnosed with microcephaly and other Zika-related illnesses; and 3) a *platform linking data from the Brazilian malaria ecosystem* (transmission vectors, notifications, and patient care) providing support for analytical methods targeting malaria eradication.

This paper proceeds as follows: Section 2 brings information on some related work. Section 3 presents our mathematical model to probabilistic record linkage. Section 4 details the implementations of our model over different parallel architectures. Section 5 presents our case studies and experimental results. Some conclusions and future work ideas close the paper.

2 Related work

Record linkage is a well-known problem that appears in data integration scenarios. The literature has a vast set of methods and tools providing from usual join and cartesian products (traditional database systems) to new approaches designed to deal with big data applications, such as NoSQL [17] and high-scalable infrastructures. Despite this apparent diversity, few proposals exploring hybrid parallel architectures exist so far.

Design and the evaluation of sequential and parallel record linkage algorithms are discussed in [12]. One approach implements a pipeline to concatenate, sort and block records, generating a graph of matching pairs. Its speedup varies from 2 to 20 with a 5 million records synthetic data set, and is 85 as many times faster than a previous Java implementation with a real data set of circa 1.1 million records. The parallel version was tested with a 6 million records synthetic data set, presenting a linear speedup varying from 7.5 to 26.4 (from 8 to 32 cores, respectively). These methods are deeper explained in [11] and a Web-based version, called RLT-S, is discussed and compared to other existing free tools (Febrl and FRIL) in [10].

A MATLAB-based parallel record linkage is presented in [9]. The authors define clean and dirty data sets, based on inclusion and symmetric difference relationships, and evaluate sequential and parallel methods over 5.000 records.

In [7], a new simhash and hamming distance implementation, targeted to identify near-duplicate documents, is discussed. The authors use public document data sets to evaluate the proposed method over CPU and GPUs architectures, obtaining speedups of up to 17% depending on the GPU utilized.

A Naïve Bayes algorithm for automatic document classification implemented over GPUs is discussed in [1]. The authors use six document databases (Medline, Reuters, Amazon etc) with variable number of attributes. The proposed method presents very similar accuracy on CPUs and GPUs and speedups varying from 12 to 35 depending on the database used.

Parallel executions of privacy-preserving linkage methods over GPUs are discussed in [19]. The authors present an OpenCL-based [14] implementation responsible for the filtering and similarity check phases of a standard record linkage pipeline. Tests were executed considering two data sets scaled up to 500.000 records, providing improvements of 10 to 20% in a hybrid scenario.

The scalability of record linkage is also discussed in [18], with an emphasis on blocking and clustering techniques. The authors use two public data sets to evaluate different ways (canopy clustering, object reduction) a scalable record linkage method can be implemented, with execution times in the magnitude of hours but with very accurate (more than 93%) results.

In [8], several algorithms for pairwise comparisons are presented and their results over NVIDIA and OpenCL GPUs, as well Intel CPUs are discussed. The authors claim a speedup of up 10 times for a dataset with 1.7 million records containing music data from freedb.org.

Table 1 summarizes existing approaches presenting some kind of parallelization on multicore or GPU environments. We compared proposals that are closest to ours in terms of: i) kind of processing unit (CPU or GPU) performing computation; ii) use of privacy-preserving techniques to anonymize identifiable attributes; iii) use of blocking strategies to reduce pairwise comparison; and iv) scalability over datasets larger than 10 millions of records.

Table 1: Parallel Approaches to Probabilistic Record Linkage.

	Characteristics					
References	GPU	CPU	Privacy-preserving	Blocking	Load balancing	Dataset >10M
Simhash [7]	✓					
Dup. detection [8]	✓			✓		
MATLAB [9]		✓				
Hierar. clustering [11]		✓		✓		
PPJoin [19]	✓		✓	✓		
Our Solution	✓	✓	✓	✓	✓	✓

To the best of our knowledge, there is no privacy-preserving record linkage able to simultaneously explore hybrid environments. Our approach contributes to the field in terms of i) the amount of data being probabilistically linked; ii) a mathematical model exploiting characteristics of hybrid parallel architectures to improve performance to huge databases; and iii) high accuracy results given the absence of gold-standards for probabilistic data linkage.

By way of complementing, it is important to notice new approaches targeting the supporting of data analytics advertised under the term "GPU databases". Among the existing approaches so far, we can mention MapD[1], Kinetica [13], SQream[2], Alenka[3], and BlazingDB[4]. All of them leverage the processing power of multiple GPUs to support machine learning, visualisation and other analytics functions over large data sets.

3 Mathematical model to record linkage

A theoretical approach to record linkage was first formalised by Fellegi and Sunter [6] and their model is currently widely accepted. Let the records in a dataset A be R_a and the records in a dataset B be R_b. A record linkage process intends to classify each pair (a, b) generated from the cross-product $A \times B$ in two resulting datasets: M (matches or true positives) and U (non-matches or true negatives). Usually, a third dataset is formed with all "dubious" records (false positives and false negatives). Since the desired cardinality for record linkage is $1:1$, meaning no duplicates are selected, the maximum size of the dataset M is equivalent to the smallest input dataset.

The probabilistic approach considers that pairwise classification is given by comparing common identifier attributes, such as name and date of birth, and calculating a similarity index to decide if the pairs match or not. Therefore, consider $\alpha(a)$ and $\beta(b)$ information inherent to a and b, respectively, a comparison function γ can be given by $\gamma(\alpha(a), \beta(b))$.

[1] https://www.mapd.com/
[2] https://sqream.com/
[3] https://github.com/antonmks/Alenka
[4] https://blazingdb.com/index.html

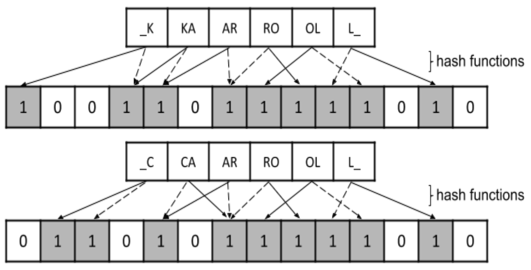

Fig. 1: Setup of Bloom filters.

Given the sensitive nature of health care data, we need to apply anonymization prior to linkage. We rely on Bloom filters [2], which are vectors of lenght l initialized with 0's. The text to be anonymized is decomposed into bigrams, i.e. character pairs including spaces. Each bigram bg, from the set $S = \{bg_1, bg_2, ..., bg_n\}$, guided by h hash functions, is mapped to a specific position between 0 and $l - 1$. The value in the mapped position is changed from 0 to 1. This stage is illustrated in Fig. 1, in which dense and dashed arrows represent different hash functions.

The linkage step itself is performed over the set of Bloom filters where a similarity measure must be calculated. We use Sørensen-Dice (or F1) score,

$$SD = \frac{2|N\_bg\_a \cap N\_bg\_b|}{N\_bg\_a + N\_bg\_b} \ , \qquad (1)$$

in which $N\_bg\_a \cap N\_bg\_b$ is the number of bits mapped for each bigram that can be found in both filters, being N_bg_a the number of bits mapped for each bigram in the first filter and N_bg_b the number of bits mapped for each bigram in the second filter. In our case, N_bg_a and N_bg_b represent the number of bits "1" in the filters. Dice calculation returns values ranging from 0 to 1, which we normalized from 0 to $10,000$.

One of the key issues in probabilistic linkage is to determine which values for SD must be used to decide if a pair matches. We should define upper and lower cut-off values to classify pairs as matches or non-matches being aware that a high value for the upper cut-off point may incur in a high number of false non-matches, whereas a low value can return lots of false matches. The overall goal is to achieve a high accuracy through the maximization of matches and the reduction of dubious records. This is a challenging task due to the absence of gold-standards and to the influence that data characteristics (quality, nature etc.) cause on the results.

4 Parallelization of probabilistic record linkage

The most critical operation within a linkage pipeline is pairwise comparison, which results in a massive usage of computational resources. Being m and n the number of records in two different databases M and N, respectively, and considering that the comparison step has complexity of $\mathcal{O}(n \times m)$, the number of iterations, as well as the size of files generated, grow quadratically. This complexity makes it difficult to use traditional technologies when the input files are very large. However, the

Algorithm 1 Sequential probabilistic linkage model.

```
INPUT
  matrixA, matrixB    /* larger and smaller matrix                             */
  nlines_a, nlines_b  /* number of lines of matrixA and matrixB                */
  num_col             /* number of columns of both matrices                    */
  bloomA, bloomB      /* bloom filter of one record from matrixA and matrixB */

OUTPUT
  /* Dice values which corresponds to the similarity between
     two records represented as bloom filters            */

cpu_exec (int *matrixA, *matrixB, nlines_a, nlines_b, num_col) {
  int *bloomA = malloc(nlines_a * num_col);
  int *bloomB = malloc(nlines_b * num_col);
  for(int i = 0; i < nlines_a; i++) {
    for (int j = 1; j < num_col; j++) {
      bloomA[j-1] = matrixA[i * num_col + j];
    }
    for (int k = 0; k < nlines_b; k++) {
      for (int l = 1; l < num_col; l++) {
        bloomB[l-1]= matrixB[k * num_col + l];
      }
      dice(bloomA, bloomB, num_col);
    }
  }
}
```

existing independency among the tasks involved in the probabilistic record linkage can help to reduce the execution time if we use parallel technics. The speedup achieved by the parallel application can be large if we are able to find the best algorithm-architecture combination for this problem.

In this paper, we evaluate the behavior of similarity calculation using the Dice index. This function considers the comparison of two records previously transformed into fixed size Bloom filters. A primary understanding of this process considers only one running thread (sequential approach). Then, we show parallel optimizations targeting multicore and multi-GPU environments.

4.1 The sequential approach to data linkage

Algorithm 1 shows the sequential approach. Function `cpu_exec` receives as arguments two matrices (`matrixA` and `matrixB`), their respective sizes (`nlines_a` and `nlines_b`) and a fixed column size corresponding to Bloom information.

A commonly strategy applied to reduce the execution time of pairwise comparison is *blocking* [20], which groups together records presenting similar values in given attributes. Although blocking helps to avoid unnecessary comparisons, they can degrade the accuracy if records referring to the same entity were mistakenly included in different groups that will never be compared. Blocking strategies should be carefully chosen, specially when dealing with large data sets. Our best sequential approach applies blocking.

Algorithm 2 Dice function for similarity calculation.

```
INPUT
  bloomA, bloomB  /* bloom filter of one record from matrixA and matrixB  */
  num_col         /* number of columns of both matrices                   */

OUTPUT
   /* Dice value which correspond to the similarity between
      two records represented as bloom filters             */

float dice(int *bloomA, int *bloomB, int num_col) {
  float a = 0, b = 0, h = 0;
  for (int i = 0; i < num_col; i++) {
    if (bloomA[i] == 1) {
      a++;
      if (bloomB[i] == 1) h++;
    }
    if (bloomB[i] == 1) b++;
  }
  float dice = (h * 2.0) / (a + b);
  return dice;
}
```

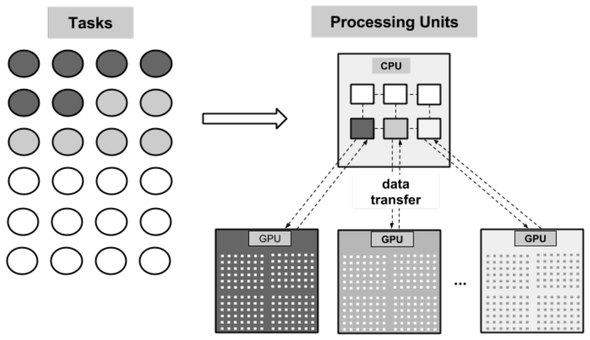

Fig. 2: General scheme of combined processing units

4.2 Using multicore and multi-GPU

The sequential approach described in Algorithm 1 has been used as a basis for the proposed parallel version for multicore and multi-GPU environments. Algorithm 1 makes use of the Dice function shown in Algorithm 2 that implements (1). As it is not hard to see we can exploit parallelism to calculate matrix elements and perform summation by partitioning the outermost loop (indexed by `i`) into independent chunks. These chunks, in turn, can be of different sizes, so that we can accommodate the workload onto a heterogeneous environment.

In a first stage, the heterogeneous parallel version is based on partitioning the workload representing the computation of elements of matrices A and B into two main sets. Each one of the two sets is simultaneously addressed over the available CPU subsystem or the GPU subsystem, respectively. We have used a static strategy to dispatch data and tasks to the available processors. As we need to have one CPU thread linked to each GPU, we initialize the runtime with as many

Algorithm 3 Using multiple GPU devices and multicore.

```
INPUT
  matrixA, matrixB        /* larger and smaller matrix                  */
  matrizA_gpu, matrizA_cpu /* part of matrixA manipulated by GPU and CPU */
  nlines_a, nlines_b      /* number of lines of matrixA  and matrixB    */
  num_col                 /* number of columns of both matrices         */
  pu_threshold            /* stores the fraction of matrixA assigned
                             to each processing unit                    */
  qtd_gpu                 /* amount of gpus available for computing     */

OUTPUT
  /* Dice values which corresponds to the similarity between
     two records represented as bloom filters             */

 1:  omp_set_nested(1);
 2:  omp_set_num_threads(num_gpus);
 3:  #pragma omp parallel num_threads(qtd_gpu+1)
 4:  {
 5:    int id = omp_get_thread_num();
 6:    if(id == 0) {
 7:      int *matrixA_cpu = split(matrixA, pu_threshold);
 8:      #pragma omp parallel num_threads(threads_cpu)
 9:      {
10:        int id_nested = omp_get_thread_num();
11:        cpu_exec(matrixA_cpu, matrixB, nlines_a, nlines_b, id_nested);
12:      }
13:    }
14:    else if(id != 0) {
15:       cudaSetDevice(id);
16:       cudaGetDevice(&gpu_id);
17:       int *matrixA_gpu;
18:       matrizA_gpu = split(matrixA, pu_threshold);
19:       kernel(matrixA_gpu, matrixB, nlines_a_gpu, nlines_b);
20:    }
21:  }
```

CPU threads as CUDA devices [3] plus a number of CPU threads linked to each CPU core. Figure 2 shows a schema of data movement represented as a model of a multicore and multi-GPU environment in which each group of processing elements executing a parallel computational kernel is seen as a combined processing unit.

Algorithm 3 shows the scheme used to distribute the workload over two computational resources. Objects `matrixA` and `matrixB` represent the input data sets containing the records to be compared. The `matrixA` is divided (`split` function at lines 7 and 18) and delivered to each CPU core while `matrixB` is fully broadcast to keep consistency. We store these matrices into arrays so each thread, identified as `thread_id`, independently whether it is linked to a CPU or a GPU, stores a chunk of `matrixA`. This chunk is calculated in `pu_threshold`. The call `cpu_exec()` performs the comparison in the same way as demonstrated in the sequential version (Algorithm 1), although some new thread management is required.

GPUs in our system are identified with integers (`gpu_id` in line 16). The first thread is bound to multicore execution, being responsible for creating the team of CPU worker threads (`threads_cpu`). The remaining threads are bound to GPUs. The core computation performed by the GPUs is implemented by the `kernel()` function and the core computation performed by the CPU cores is in `cpu_exec()`.

Computation performed by GPUs is implemented in the `kernel()` function, called in line 19 of Algorithm 3. This function firstly performs the typical operations of uploading data from the CPU to the GPU prior to call the CUDA kernel. Thus, in order to execute this kernel, it is supposed that `matrixA`, `matrixB`, `nlines_a` and `nlines_b` have been previously uploaded into the GPU memory.

The problem to be solved through the kernel is highly parallelizable. All elements from `matrixA` and `matrixB` can be computed concurrently. We need to imagine the shared data as a one-dimensional cube where each position has a partial comparison of each element of `matrixA` and `matrixB`. In other words, elements `matrixA`, and `matrixB`, for all `i`, contain the partial comparison corresponding to a given element, taking into account the parity between the array element and the thread coordinates. After this, it is necessary to add all partial comparisons.

However, the total amount of shared memory is what really determines the size of thread blocks. Anyway, the limitation in the number of threads per block is easily overcome by the number of blocks that can be run concurrently. More blocks means that data computed by each block in that dimension and stored in the shared memory should be also shared among the thread blocks. This can only be done through global memory yielding in a performance penalty.

5 Experimental Results

Our execution board comprises 4 Intel Xeon processors at 3.33 GHz and 100 GB DDR3 main memory. Each processor has 6 cores with 12 MB of cache memory. The board also contains 2 NVIDIA Tesla C2070 GPUs featuring each GPU 14 multiprocessors (SM) with 32 stream processors (SP) each (448 cores in total). Floating point operations follow the IEEE 754-2008 standard. It is expected up to 515 Gigaflops of double-precision peak performance in each GPU. The installed CUDA toolkit is version 7.5.

In our experiments we use a parallel version of Algorithm 3 implemented using OpenMP [16] and CUDA [15] for the probabilistic linkage model used in our heterogeneous environment. Many parameter values were used at installation time to estimate the best values for system parameters. The available range for CPU cores (c) is $1, 2, \ldots, 32$, with Intel Hyper-Threading [5] set. Then, we checked for GPUs workloads (w) from 10% to 45%. The input sizes of the problem and the number of records (s) for the experiments were $1,000,000$, $2,000,000$, $\ldots$, $20,000,000$. Table 2 shows execution times of different values of system parameters and emphasize in bold those that are the best ones.

There are two important observations: (1) c values depend on the problem size (s) in the system under test; and (2) for each problem size and for different values of w, we obtain a different optimum value for c on each execution environment. Be aware of this variability is essential to make good decisions in the selection of optimum algorithm parameters. We did experiments with different combinations of s, c, and w, considering the problem size to obtain the model on a given platform.

Due to the high time required for the sequential execution, and even to the parallel run using only the CPU cores, we also apply a standard blocking strategy before the comparison step begins. These results, shown in Fig. 3, are useful for performance comparison in subsequent steps. The execution time with multiple threads is denoted by "CPU cores". On the CPU side, the calculation data mart

Table 2: Execution times obtained with different values for the performance parameters (the best values are marked in boldface).

	$w = 45, 45, 10$		$w = 40, 40, 20$		$w = 35, 35, 30$	
s	c	$t(s, c, w)$	c	$t(s, c, w)$	c	$t(s, c, w)$
1,000,000	16	4.21	16	4.24	**24**	**4.14**
2,000,000	16	4.67	16	4.60	**24**	**4.58**
4,000,000	16	5.62	24	5.52	**28**	**5.43**
6,000,000	24	6.64	24	6.44	**32**	**6.23**
8,000,000	24	7.53	24	7.42	**32**	**7.36**
10,000,000	24	8.13	24	8.14	**32**	**8.01**
12,000,000	24	8.76	32	8.79	**32**	**8.35**
14,000,000	32	9.81	32	9.28	**32**	**9.33**
16,000,000	32	10.31	32	10.33	**32**	**10.54**
18,000,000	32	11.47	32	11.04	**32**	**10.64**
20,000,000	32	12.91	32	12.05	**32**	**11.52**

is distributed among threads and each thread runs exclusively on a CPU core. Versions denoted by "1 GPU" and "2 GPUs" represent executions in one device and two devices, respectively.

The heterogeneous model ("Hybrid") uses all cores available in the heterogeneous system. In this model, the threads are executed by all the computing elements in the machine with the suitable number of CPU cores and the two GPUs. The results show that the parallel CPU algorithm reduces the execution time significantly, as can be observed in Fig. 3.

We show in Fig. 3(a), Fig. 3(b) and Fig. 3(c), the execution time and the speedup, respectively, for the probabilistic linkage algorithms with different sizes ranging from 1,000,000 to 20,000,000. The execution was carried out on each subsystem independently (CPU cores, 1GPU, 2GPUs and Hybrid) to have a measure for comparison purposes.

Speedup has been obtained with regard to the use of the CPU cores subsystem using a blocking strategy. As can be seen in Fig. 3(c), the maximum speedup is around 8 with the hybrid subsystem, presenting a difference in performance that can be observed more clearly. Both plots show how the use of GPUs in our system clearly outperforms the computation on the CPU cores.

We show different aspects of the behavior of the algorithm with different workloads. Figure 3(d) plots the evolution of time regarding identical GPUs. Based on the experiments, it can be observed that the value w obtained through the theoretical derivation is the best to be chosen if we use 2 GPUs and all the CPU cores. This experiment shows the reduction in time achieved by the use of 2 GPUs and the CPU cores, and how this improvement grows with the problem size due to the parallelization of the probabilistic linkage. As expected, the number of records does not represent a big difference for different workloads due to the small weight of communications (CPU-GPU) compared to the weight of computations. The best behavior is around (w) = (GPU, GPU, CPU) = (35%, 35%, 30%) for the system under test.

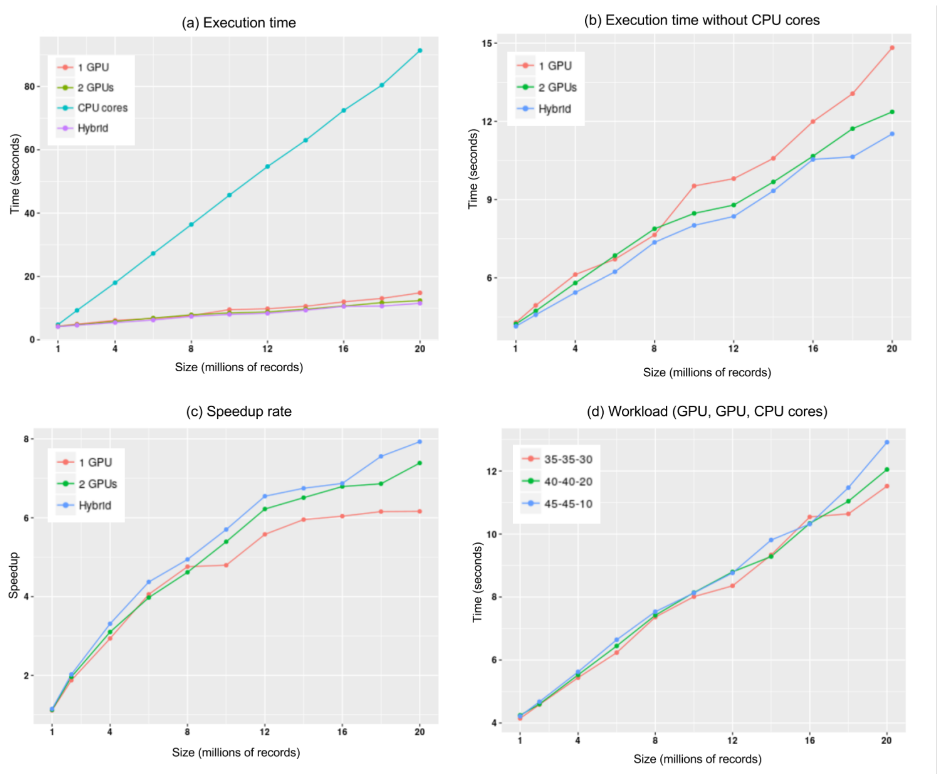

Fig. 3: Performance of probabilistic record linkage algorithms by varying the problem size. (a) Execution time. (b) Execution time without CPU cores. (c) Speedup rate. (d) Workload distribution.

6 Conclusions and Future Directions

The parallel algorithm developed in this project enables to efficiently compute the core computation that appears in the problem of probabilistic record linkage. We have shown how it is possible to solve current computational problems associated with the problem of record linkage. To show this, we have addressed a particular case in the context epidemiological studies that involve the management of very large Brazilian governmental datasets, what results in substantial benefits for the public health in this country. The solution proposed is simple and efficient, allowing to exploit the heterogeneous nature of the underlying computational resources. Our model of implementation and workload distribution is portable, so it can be used on many configurations including many multicore CPUs, NVIDIA GPUs, even processors based on the Intel MIC architecture.

There exists, however, an important issue to address in the future related to the difficulty of management of very large datasets. The large amount of memory required by the process is currently the hurdle we are facing to scale the proposed work to databases with more than 100 million records.

Acknowledgments

This work has been partially supported by CNPq, FAPESB, Bill & Melinda Gates Foundation, The Royal Society (UK), Medical Research Council (UK), NVIDIA Hardware Grant Program, Generalitat Valenciana (grant PROMETEOII/2014/003), Spanish Government and European Commission through TEC2015-67387-C4-1-R (MINECO/FEDER), and network CAPAP-H. We have also worked in cooperation with the EU-COST Programme Action IC1305, "Network for Sustainable Ultrascale Computing (NESUS)".

References

1. Andrade, G., Viegas, F., Ramos, G.S., Almeida, J., Rocha, L., Gonçalves, M., Ferreira, R.: GPU-NB: A fast CUDA-based implementation of Naïve Bayes. In: 2013 25th International Symposium on Computer Architecture and High Performance Computing, pp. 168–175 (2013)
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
3. Cook, S.: CUDA Programming: A Developer's Guide to Parallel Computing with GPUs, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2013)
4. Doan, A., Halevy, A., Ives, Z.: Principles of Data Integration. Elsevier Science (2012)
5. Étienne, E.Y.: Hyper-threading. TurbsPublishing (2012)
6. Fellegi, I.P., Sunter, A.B.: A theory for record linkage. Journal of the American Statistical Association **64**, 1183–1210 (1969)
7. Feng, X., Jin, H., Zheng, R., Zhu, L.: Near-duplicate detection using GPU-based simhash scheme. In: 2014 International Conference on Smart Computing, pp. 223–228 (2014)
8. Forchhammer, B., Papenbrock, T., Stening, T., Viehmeier, S., Naumann, U.D.F.: Duplicate detection on GPUs. In: BTW, pp. 165–184. Köllen-Verlag (2013)
9. Kim, H.s., Lee, D.: Parallel linkage. In: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management, CIKM 2007, pp. 283–292. ACM, New York, NY, USA (2007)
10. Mamun, A.A., Aseltine, R., Rajasekaran, S.: RLT-S: A web system for record linkage. PLOS ONE **10**(5), 1–9 (2015)
11. Mamun, A.A., Aseltine, R., Rajasekaran, S.: Efficient record linkage algorithms using complete linkage clustering. PLOS ONE **11**(4), 1–21 (2016)
12. Mamun, A.A., Mi, T., Aseltine, R., Rajasekaran, S.: Efficient sequential and parallel algorithms for record linkage. Journal of the American Medical Informatics Association **21**(2), 252–262 (2014)
13. Mizell, E., Biery, R.: How GPUs are defining the future of data analytics (2017)
14. Munshi, A., Gaster, B., Mattson, T.G., Fung, J., Ginsburg, D.: OpenCL Programming Guide, 1 edn. Addison-Wesley Professional (2011)
15. NVIDIA Corporation: NVIDIA CUDA C programming guide (2010). Version 3.2
16. OpenMP Architecture Review Board: OpenMP application program interface version 4.0 (2013)
17. Pokorny, J.: NoSQL databases: A step to database scalability in web environment. In: Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services, iiWAS '11, pp. 278–283. ACM, New York, NY, USA (2011)
18. Rendle, S., Schmidt-Thieme, L.: Scaling Record Linkage to Non-uniform Distributed Class Sizes, pp. 308–319. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
19. Sehili, Z., Kolb, L., Borgs, C., Schnell, R., Rahm, E.: Privacy preserving record linkage with ppjoin. In: Datenbanksysteme für Business, Technologie und Web (BTW), pp. 85–104 (2015)
20. Winkler, W.E.: The state of record linkage and current research problems (1999)
21. Zhong, Z., Rychkov, V., Lastovetsky, A.: Data partitioning on multicore and multi-GPU platforms using functional performance models. IEEE Transactions on Computers **64**(9), 2506–2518 (2015)