



Federal University of Bahia
Institute of Mathematics and Statistics

Graduate Program in Computer Science

**EXPLOITING HETEROGENEOUS
COMPUTING TECHNIQUES TO ADDRESS
PROBABILISTIC BIG DATA LINKAGE**

Clícia dos Santos Pinto

DOCTORAL THESIS

Salvador, Bahia - Brazil
28 July 2020

CLÍCIA DOS SANTOS PINTO

**EXPLOITING HETEROGENEOUS COMPUTING TECHNIQUES
TO ADDRESS PROBABILISTIC BIG DATA LINKAGE**

This PhD thesis was presented to the Graduate Program in Computer Science (PGCOMP) of the Federal University of Bahia (UFBA) as a partial requirement for the granting of the title of Doctor (PhD) in Computer Science.

Advisor: Prof. Dr. Marcos Ennes Barreto

Salvador, Bahia - Brazil
28 July 2020

Sistema de Bibliotecas - UFBA

Pinto, Clícia.

Exploiting heterogeneous computing techniques to address probabilistic big data linkage / Clícia dos Santos Pinto – Salvador, Bahia - Brazil, 2020. 88p.: il.

Advisor: Prof. Dr. Prof. Dr. Marcos Ennes Barreto.

Tese de (Doutorado) – Federal University of Bahia, Institute of Mathematics and Statistics, 2020.

1. Data linkage. 2. Heterogeneous computing. 3. Optimization. 4. Graphic processors. I. Barreto, Marcos E.. II. Federal University of Bahia. Institute of Mathematics and Statistics. III Título.

CDD – XXX.XX

CDU – XXX.XX.XXX

CLÍCIA DOS SANTOS PINTO

**“EXPLOITING HETEROGENEOUS COMPUTING TECHNIQUES TO ADDRESS
PROBABILISTIC BIG DATA LINKAGE”**

Esta tese foi julgada adequada à obtenção do título de Doutor em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da UFBA.

Salvador, 28 de julho de 2020



Prof. Dr. Marcos Ennes Barreto
(Orientador - PGCOMP/UFBA)



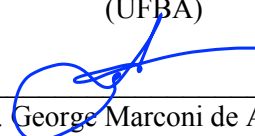
Prof. Dr. Rodrigo da Rosa Righi
(UNISINOS)



Prof. Dr. Esbel Tomás Valero Orellana
(UESC)



Prof. Dr. Maycon Leone Maciel Peixoto
(UFBA)



Prof. Dr. George Marconi de Araújo Lima
(UFBA)

To my sister, that supports me since the first steps of this journey and whose strength and competence have always been an example and an inspiration, I dedicate this work and everything it means.

ACKNOWLEDGEMENTS

I would like to express my thankfulness to Professor Marcos Barreto, mainly for his confidence and direction during this research development and during my academic formation. His knowledge that goes beyond technical competence helped me to keep the focus on the development of proposed approaches and, at the same time, to place this research in a local and global social context. The dynamics he built AtyImo-Lab are a great reference and object of enormous admiration.

With a heart now full of joy, I thank my partner Anderson Amorim for share the best and worst implications of this research conduction. I appreciate your patience, perspective, trust, and continuous support. Sharing life with you makes everything easier.

To my parents, my sincere gratitude for helping me to get this far. I am also grateful to my siblings and nephew for the friendship, care, presence, and encouragement many times along the way.

RESUMO

Embora a computação heterogênea seja uma poderosa abordagem para a resolução de problemas computacionalmente intensivos, o seu desempenho e eficiência estão profundamente atrelados às propriedades da carga de trabalho a que são submetidos. O gerenciamento de grandes volumes de dados em ambientes heterogêneos implica na escolha de algoritmos dinâmicos de escalonamento e particionamento que minimizem o tempo de resposta e o volume de comunicação entre as unidades de processamento, ao mesmo tempo em que assegurem escalabilidade. Esta exigência tem se tornado mais urgente à medida que os dispositivos que compõem as plataformas heterogêneas se tornam mais numerosos e diversificados. Este trabalho apresenta uma metodologia para a exploração de técnicas de computação heterogênea em ambientes compostos por CPUs e GPUs para aplicações de vinculação probabilística de grandes volumes de dados, bem como propõe a integração deste método à ferramenta AtyImo, desenvolvida parcialmente durante esta pesquisa. A metodologia proposta permite uma distribuição de dados e tarefas adequada às aplicações que manipulam grandes conjuntos de dados, mais especificamente aplicações de vinculação de registros (*data linkage*). Como prova de conceito, a solução implementada foi utilizada para integrar dados socioeconômicos em larga escala (100 milhões de registros) com dados de saúde pública armazenados em diferentes fontes governamentais brasileiras. Através da metodologia proposta foi possível vincular 1×10^{12} pares de registros em um tempo total próximo a uma hora, o que pode ser considerado um resultado promissor em relação às ferramentas de vinculação de dados existentes. Estes resultados demonstram que a solução desenvolvida possui bom desempenho e se apresenta como alternativa viável para resolver problemas comuns de escalabilidade relacionados à vinculação de registros. A possibilidade de vinculação probabilística de grandes volumes de dados sobre arquiteturas híbridas, explorando a natureza heterogênea dos recursos disponíveis e com tempo de execução extremamente eficiente, constituem as principais contribuições deste trabalho.

Palavras-chave: Vinculação de dados. Balanceamento de carga. Computação paralela heterogênea. Aceleradores gráficos.

ABSTRACT

Although heterogeneous computing is a powerful approach to solve computationally intensive problems, its performance and efficiency highly depend on the workload to which they are exposed. Managing large volumes of data in heterogeneous environments involves choosing efficient scheduling and partitioning algorithms that minimize the response time and the volume of communication among processing units while ensuring scalability. This requirement has become more urgent as the devices composing such heterogeneous platforms become more numerous and diversified. This work presents a methodology for using heterogeneous computing techniques over hybrid CPU+GPU environments to allow for data and task distribution within big data linkage applications. This methodology was integrated into the AtyImo tool, which was partially developed during this research to provide probabilistic record linkage. As proof of concept, the implemented solution was used to integrate a large-scale (100 million records) socioeconomic database with public health data from disparate governmental sources. The proposed methodology is able to perform 1×10^{12} pairwise comparison in around one hour, which is a quite prominent result amongst existing data linkage tools. Observed results evidence that the developed solution achieves good performance and can be an alternative to solve scalability issues in data linkage contexts. The possibility of probabilistically linking massive datasets using hybrid architectures and exploring the heterogeneous nature of available resources with an efficient execution time are the main contributions of this work.

Keywords: Data linkage. Load balancing. Heterogeneous parallel computing. Graphical accelerators.

CONTENTS

| | |
|---|----|
| Chapter 1—Introduction | 1 |
| 1.1 Contextualization | 1 |
| 1.2 Motivation | 3 |
| 1.3 Objectives | 4 |
| 1.4 Research Questions | 5 |
| 1.5 Technical Challenge | 5 |
| 1.6 Research Method | 5 |
| 1.6.1 Step 1 – Review of parallel probabilistic record linkage concepts | 6 |
| 1.6.2 Step 2 – Revision of the AtyImo data linkage tool | 6 |
| 1.6.3 Step 3 – Parallel and heterogeneous modelling | 6 |
| 1.6.4 Step 4 – Performance and accuracy evaluation | 6 |
| 1.7 Datasets | 6 |
| 1.8 Metrics and Experimental Setup | 6 |
| 1.9 Contributions | 7 |
| Chapter 2—Related Work | 9 |
| Chapter 3—Literature Review | 11 |
| 3.1 Heterogeneous Computing Platforms | 11 |
| 3.1.1 Characterization of heterogeneous platforms | 12 |
| 3.2 Algorithmic Issues concerning Heterogeneous Computing Platforms | 15 |
| 3.2.1 Workload-partitioning techniques over heterogeneous platforms | 15 |
| 3.3 Big Data Processing and Analysis | 17 |
| 3.4 Data Integration in the Context of Big Data | 19 |
| 3.5 Formulation of the Record Linkage Problem | 21 |
| 3.5.1 Privacy preservation in probabilistic record linkage | 21 |
| 3.5.2 Indexing and blocking construction | 23 |
| 3.5.3 Similarity calculation | 24 |
| Chapter 4—Case Study: Integrating Large Governmental Databases | 27 |
| 4.1 Longitudinal Studies Based on Population Cohorts | 27 |
| 4.2 Brazillian Governmental Databases | 28 |
| 4.3 The 100 Million Brazilian Cohort | 30 |

| | |
|---|-----------|
| Chapter 5—AtyImo-H Data Linkage Tool | 33 |
| 5.1 AtyImo-H | 33 |
| 5.2 Data Integration Model | 34 |
| 5.3 AtyImo-H Record Linkage Pipeline | 36 |
| 5.3.1 Application pipeline details | 37 |
| 5.3.1.1 Pre-processing | 37 |
| 5.3.1.2 Transformation | 38 |
| 5.3.1.3 Comparison | 38 |
| 5.3.1.4 Sorting | 38 |
| 5.3.1.5 Record retrieval | 38 |
| Chapter 6—Parallel Modeling for Record Linkage | 41 |
| 6.1 Probabilistic Record Linkage | 41 |
| 6.1.1 Cross-comparison | 43 |
| 6.1.2 Approximate similarity measurement | 44 |
| 6.2 Heterogeneous Modelling | 45 |
| 6.2.1 Linkage pipeline adapted to a hybrid model | 45 |
| 6.2.2 Data load and granularity | 46 |
| 6.2.3 Data partitioning | 48 |
| 6.2.4 Multicore parallel probabilistic comparison | 50 |
| 6.2.5 Hybrid-parallel probabilistic comparison over heterogeneous environment | 52 |
| 6.3 Discussion concerning technologies | 54 |
| Chapter 7—Hybrid Record Linkage Experiments | 55 |
| 7.1 Accuracy Analysis | 55 |
| 7.2 Performance Analysis | 56 |
| 7.2.1 Hybrid record linkage up to 20 million records | 58 |
| 7.2.1.1 Experimental setup (test purpose) | 58 |
| 7.2.1.2 Performance analysis | 58 |
| 7.2.2 Hybrid record linkage up to 70 million records | 61 |
| 7.2.2.1 Experimental setup | 61 |
| 7.2.2.2 Performance analysis | 62 |
| 7.2.3 Hybrid record linkage up to 100 million records | 68 |
| 7.2.3.1 Experimental setup | 68 |
| 7.2.3.2 Performance analysis | 69 |
| 7.2.4 Comparison with Spark-based AtyImo | 75 |
| Chapter 8—Conclusion | 79 |
| 8.1 Final Considerations | 79 |
| 8.2 Future Work | 81 |

LIST OF FIGURES

| | | |
|-----|---|----|
| 3.1 | Hybrid architecture with different interconnection links and PUs (CPUs and GPUs). | 13 |
| 3.2 | Gap between ideal and actual ability to analyze big data (LEMIEUX; GORMLY; ROWLEDGE, 2014). | 18 |
| 3.3 | Generic big data pipeline and knowledge discovery process. | 18 |
| 3.4 | Pairwise comparison of two records (A and B) containing deterministic (exact) and probabilistic (approximate) attributes. | 20 |
| 3.5 | Example of Bloom filters: applying hash functions to bigrams of a string. | 22 |
| 3.6 | Example of a blocking predicate implemented in this work. | 24 |
| | | |
| 5.1 | Input and output of the three main modules of AtyImo-H’s pipeline. | 35 |
| 5.2 | Execution pipeline for probabilistic record linkage. | 37 |
| 5.3 | Execution pipeline for hybrid model that performs probabilistic linkage on heterogeneous platforms. | 39 |
| | | |
| 6.1 | High-level overview of parallel linkage and deduplication using 3 Processing Units (one CPU and two GPUs). a) Data partitioning considering deduplication been performed on CPU. b) Data partitioning considering 2/3 of deduplication been performed by GPU. | 49 |
| 6.2 | Data transfer scheme considering available PUs on the heterogeneous environment. | 50 |
| | | |
| 7.1 | Accuracy assessment in the integration of SIM-SE and CADU (cohort) data. | 56 |
| 7.2 | Accuracy assessment in the integration of SIM-SC and CADU (cohort) data. | 57 |
| 7.3 | Accuracy assessment in the integration of SIM-RO and CADU (cohort) data. | 57 |
| 7.4 | Execution time for linkage step considering CPU, one GPU and two GPUs executed in isolation and its comparison with the hybrid approach. | 60 |
| 7.5 | Execution time for linkage step considering one GPU and two GPUs executed in isolation and its comparison with the hybrid approach. | 60 |
| 7.6 | Speedup for the linkage step considering each approach implemented. | 61 |
| 7.7 | Execution time for each workload division between PUs in the sequence: GPU, GPU, and CPU. | 61 |
| 7.8 | Execution time of the hybrid approach and its comparison with CPU and GPU runtime in an environment composed of two P100 GPUs. | 62 |
| 7.9 | Performance gain of multi-GPU execution in comparison with CPU execution time. | 64 |

| | | |
|------|---|----|
| 7.10 | Execution time considering only multi-GPU execution, highlighting for each of them a) complete execution time, b) data transfer time, and c) kernel execution time, in an environment composed of two P100 GPUs. | 64 |
| 7.11 | Kernel execution time of both GPUs in an environment composed of two P100 GPUs. | 65 |
| 7.12 | Strong scaling given by the execution time for two different input sizes in an environment composed of two P100 GPUs. | 66 |
| 7.13 | Strong scaling given by the speedup factor up to 80 threads for two different input sizes in an environment composed of two P100 GPUs. | 66 |
| 7.14 | Strong scaling given by the speedup factor up to 40 threads for two different input sizes in an environment composed of two P100 GPUs. | 67 |
| 7.15 | Weak scaling given by the execution time up to 64 threads and 64 million records in an environment composed of two P100 GPUs. | 68 |
| 7.16 | Execution time of hybrid approach and its comparison with CPU and GPU runtime in an environment composed of two V100 GPUs. | 69 |
| 7.17 | Performance gain of multi-GPU execution in comparison with CPU execution time in an environment composed of two P100 GPUs. | 71 |
| 7.18 | Execution time considering only multi-GPU execution, highlighting for each of them: a) complete execution time, b) data transfer time, and c) kernel execution time, in an environment composed of two V100 GPUs. | 72 |
| 7.19 | Kernel execution time of both GPUs in a environment composed of two V100 GPUs. | 72 |
| 7.20 | Strong scaling given by the execution time for two different input sizes in a environment composed of two V100 GPUs. | 73 |
| 7.21 | Strong scaling given by the speedup factor up to 80 threads for two different input sizes in a environment composed of two V100 GPUs. | 73 |
| 7.22 | Strong scaling given by the speedup factor up to 40 threads for two different input sizes in a environment composed of two V100 GPUs. | 74 |
| 7.23 | Weak scaling given by the execution time up to 64 threads and 64 million records in a environment composed of two V100 GPUs. | 74 |
| 7.24 | AtyImo-H CPU X GPU-based execution (pairwise comparison step only). | 76 |

LIST OF TABLES

| | | |
|-----|---|----|
| 3.1 | Advantages and disadvantages of a hybrid parallel application running on a heterogeneous platform composed of multicore and manycore PUs. . . . | 15 |
| 3.2 | Qualitative difference between volume, velocity and variety of generated data and ability to analyze it over time (Adapted from (KAY; HARMELEN, 2014)). | 18 |
| 4.1 | Government databases used in this research. | 29 |
| 5.1 | Availability of data cleansing functionality across different linkage tools. . | 36 |
| 7.1 | Execution time for different performance parameters (best values in bold). | 58 |
| 7.2 | Comparison of execution times (in seconds) (best values in bold). | 59 |
| 7.3 | Comparative performance, given by the gain with respect to time of multicore execution (best values in bold). | 59 |
| 7.4 | Execution time of each PU and time spent loading datasets into devices' memory up to 70 million records (time in seconds. | 63 |
| 7.5 | Execution time of each PU and time spent loading datasets into devices' memory until 100 million records (time in seconds). | 70 |
| 7.6 | Input data sizes loaded into each Processing Unit for different numbers of records. | 75 |

This chapter introduces the topic of this PhD. research, presenting an overview of our motivation, objectives, and the research questions that was pursued. Additionally, the research method employed during this investigation is also described.

INTRODUCTION

1.1 CONTEXTUALIZATION

In recent years, we have faced the exponential growth of personal and corporate data. Following this, the demand for scalable and efficient processing and analysis strategies has also increased. Often both data- and compute-intensive problems that examine, analyze and process very large scale datasets and are naturally parallelizable, increasingly use parallel platforms and several paradigms that propose efficient distribution, processing, and communication to deal with the big volumes of data in scenarios involving intensive simulation, modeling, data mining, and other commercial and scientific applications. Combining high-performance algorithms with efficient hardware that provides high bandwidth access, storage and powerful processing capabilities can circumvent technological limitations to accomplish this need.

A specific data-intensive problem applied to this context arises from the need of linking data from disparate datasets. This class of data integration tasks aims at to find data from two or more records representing the same real-world entity which are usually stored in disparate sources (DOAN; HALEVY; IVES, 2012). When deterministic record linkage is not possible due to the absence of common key attributes, probabilistic strategies should be applied. For this purpose, a comparison approach and a function for calculating similarity should be established to define an approximation degree between examined information and to classify all pairs into specific affinity groups.

The record linkage process involves several steps: data acquisition and organization, descriptive analysis, data quality assessment, standardization and cleansing, anonymization (for sensitive data), block building (or indexing) for reducing the dimensionality search space during pairwise comparison, and finally the comparison itself and similarity analysis (to decide whether or not the records belong to the same entity).

Assuming that traditional technologies are not necessarily adequate or customizable to address the immense volume of data that characterizes data-intensive computing, some

technologies are presented as an alternative to meet these needs. The complexity of the comparison step and the large volume of real datasets are major motivators for developing parallel solutions to perform probabilistic record linkage. Deciding which approach ensures better efficiency and performance for this process generally involves several factors such as response time, type of workload, communication, data granularity, and issues related to available hardware resources. Details such as input and output bottleneck caused by the tendency to use centralized storage, data transfer and management/distribution of tasks should be taken into account. The evolution of large-scale parallel and distributed systems has made it possible to support intensive computing, data distribution, and resource management.

It is now clear that data-intensive problems of this class are benefiting from heterogeneous hardware composition made up of processors and coprocessors to accelerate the computation. For a long time, graphics cards are no longer exclusively dedicated to real-time rendering applications and became able to accelerate general-purpose high-performance computing applications as well (BOLZ et al., 2003) (FUNG; TANG; MANN, 2002). Since then, several scientific applications have benefited from this composition to mainly reduce execution time and generate timely responses. Such computational accelerators are built on different architectures and programming models, requiring an agile strategy for data partitioning and task scheduling (ZHONG; RYCHKOV; LASTOVETSKY, 2015).

Hybrid models have been used to improve applications performance by increasing parallel processing capabilities and optimizing the use of available resources by gathering devices with different architectures and enabling instructions to be executed in the architecture that best fits the requirements of the applications. The fact is that there are several approaches providing data and task parallelization in a heterogeneous environment. Also, there are many optimization models and workload distribution, so each approach offers advantages in specific contexts. Besides, in heterogeneous computing platforms, data partitioning and distribution strategy are largely responsible for the performance of parallel applications. In scenarios where the volume of input data is large, new challenges are introduced as optimal distribution is limited by the memory capacity of the devices and the overhead imposed by data transfer, among other factors. Therefore, another important challenge is the development of a strategy able to hide the latency caused by the transfer of a large volume of data over the common interconnection bus.

The case study used as proof of concept and validation in this work is a data-intensive application dedicated to perform a probabilistic record linkage of different governmental datasets. Considering the high complexity of the steps composing this problem, many researches have employed strategies to improve the overall performance and efficiency of this kind of application. Compared to a simple dataset join, probabilistic record linkage has a significant problem to be considered: once two records from two different datasets (A and B) are matched and merged to C, C needs to be compared to the rest of records from the original datasets A and B again since it may incur new correspondences (KIM; LEE, 2007). Blocking (or indexing) methods, for example, propose to reduce the search space by making less pairwise comparison. The quadratic comparison in addition to the size of the input problem is prohibitive for any sequential execution, and parallel techniques

are required to generate timely responses useful for decision making, for example.

Many existing research considers the use of parallel and distributed computing platforms to solve the probabilistic record linkage problem. Most of these studies are intended to evaluate application performance in terms of execution time and accuracy, proposing improvements that consider execution over distributed big data frameworks. Despite all the efforts, few approaches consider the coordinated use of CPU and graphics accelerators (GPUs) to solve this kind of problem efficiently and effectively.

The focus of this work was the systematic experimentation and investigation of heterogeneous computation techniques in environments composed of multicore processors and multi-GPUs in order to accelerate the most costly steps of the probabilistic record integration problem. We also aimed at to develop a solution that could be integrated into the AtyImo tool (PITA et al., 2018). Specifically, the step that establishes a comparison between all candidate pairs and calculates a similarity index for each of them to decide whether the pairs belong to the same entity (*match*) or not (*non-match*) is the main module considered for parallel optimization, since it corresponds to the majority of the total execution time.

1.2 MOTIVATION

The recent rise and popularity of big data processing and analysis methodologies have led the community to develop framework-oriented solutions that take advantage of an easier implementation, transparency of parallel computing methods and fault tolerance strategies avoiding data loss by eliminating the single points of failures. While this accelerates the development of algorithms and the construction of analytical solutions, it imposes a limitation for programmers in adapting the parallel solution to the specificities of the underlying hardware.

Due to the new scenario imposed by the consolidation of big data and also the new challenges that came with the exascale computing era, new research has been devoted to investigating issues related to the integration of high-performance computing strategies with big data needs. This new effort is also due to the fact that much of the data that has emerged comes from scientific and industrial applications, forcing the high-performance computing (HPC) community to keep up with these changes. To ensure efficient analysis of data as large as those considered here, the hardware platform must be scalable and the choice of the platform type becomes a crucial decision step. The large-scale employment of heterogeneous computing techniques to accelerate high-performance applications relies primarily on the expectation for reducing execution time without neglecting management and cost issues.

The development of a more efficient solution to provide parallel and distributed computation for data-intensive application is important to improve the quality of stored data that will be used for future analysis since it is possible to test and experiment more pre-processing models when building the data warehouse and also to improve performance and generate responses in a reasonable time. Another advantage for the use of high-performance solutions applied to big data scenarios is the possibility of having a cost-effective pipeline since less time will be spent by analysts (human resources). Besides,

heterogeneous computing approaches can provide less idleness of processing unities and other hardware components, ensuring energy saving. In this sense, we sought to make maximum use of the multiple computing resources (processing units) available in the platform rather than investing exclusively in improving CPU performance. With the growing popularity of GPUs as accelerators for scientific applications, this specialized hardware is becoming more affordable. Due to the positive cost-effectiveness, many laboratories and computing-intensive/HPC centers have been equipped with such accelerators.

The configuration of today’s accelerated architectures can be seen as a differential in generating efficient models for processing and analyzing large volumes of data and performing parallel computation on them. The recent propensity to apply heterogeneous computing techniques to data-intensive problems has encouraged several researches in this regard. Specifically, the vast majority of current efforts seek methods of combining the use of such coprocessors to accelerate machine learning algorithms (CATANZARO; SUNDARAM; KEUTZER, 2008)(CHEN et al., 2014).

The focus of this research is not only on studying methods for analyzing the performance of GPU-accelerated algorithms but also on the parallel optimization of the most costly steps of data-intensive problems, where probabilistic record linking is included. Despite all recent research efforts, few studies have been directed at heterogeneous performance analysis, parallel optimization, and scalability of such problems, since historically the cost of data transfers has always limited the capacity of processing large datasets.

This research was firstly motivated by the limitation of current references devoted to the in-depth study of partitioning techniques and load balancing applied to data-intensive applications on hybrid platforms.

1.3 OBJECTIVES

Considering the presented questions and the research opportunities associated with them, the overarching goal of this work is the study and experimentation of heterogeneous computation techniques in environments composed of multicore (CPUs) and manycore (GPUs) processing units. Besides that, we also intended to examine models for data partitioning and dynamic task scheduling among different processing units to support applications with large workloads. Our specific research goals were as follows:

Research Goal 1: Ensure the scalability of the comparison step of big data linkage applications by providing a high-performance alternative that can be integrated into the AtyImo tool.

Research Goal 2: Propose an approach to ensure efficient data partitioning among different processing units, as well as optimal resource allocation for tasks belonging to probabilistic record linkage.

Research Goal 3: Contribute to the understanding of performance limitations in heterogeneous architectures when scaling a big data application over GPU-accelerated platforms.

1.4 RESEARCH QUESTIONS

During this research, we aimed at to answer two research questions:

- A. *Can data-intensive applications related to big data integration benefit from using a hybrid approach that considers graphics processing units to efficiently speedup their time-demanding steps?*
- B. *The use of a native language and parallel programming paradigm specific to high-performance applications will allow more efficient control of hardware resources so that the performance improvements will be sufficient to compensate data transfer overheads and limitations related to managing a massive workload in the memory of different devices?*

As we describe in the development of this work, using hybrid parallel architectures enables better scalability and performance control for tasks involving the linkage of large datasets. We show how to hide latency overhead caused by data transfer and we also specify the limitations of keeping a massive workload at the devices' main memory.

1.5 TECHNICAL CHALLENGE

Probabilistic record linkage, as well as most data integration problems, requires that each record of a given dataset be compared with all records of the other dataset and a similarity score be calculated. Since no duplicates exist, the comparison result for one pair needs to be kept in the device's memory until a higher similarity pair is identified. The comparison that achieves the highest agreement stores the similarity value and the indexes for retrieving each full record that makes up the pair. In large datasets, the memory needed to store these cross join results becomes a limiting factor. The need to perform intermediate disk I/O operations is a major disadvantage to the use of some big data parallel frameworks, such as Hadoop (SHVACHKO et al., 2010).

In contrast, the effort to keep large amounts of data in device's memory faces restrictions related to hardware capacity. Graphic cards, for example, traditionally have restricted storage capacity, requiring alternative methods to ensure maximum use of its global memory space. This was the first technical challenge faced in this research.

Further, it is necessary to consider the latency caused by data transfers. Accelerators are traditionally connected to the host by the PCI-e bus through which data travels at a speed that is not equivalent to the performance expected by the heterogeneous application. To work around this problem, asynchronous transfer and algorithmic optimization practices were employed to prevent overall application performance from being degraded.

1.6 RESEARCH METHOD

This section presents the methodology employed for conducting this research. We split our work into four steps as follows.

1.6.1 Step 1 – Review of parallel probabilistic record linkage concepts

This step sought to provide a literature review and a deepen knowledge of the state-of-the-art for parallel methods. To accomplish this, we investigated the current data linkage method applied by the Brazillian government and defined the limitations of their solution. We also performed an exhausting literature review to identify approaches similar to ours.

1.6.2 Step 2 – Revision of the AtyImo data linkage tool

This step was dedicated to reviewing AtyImo, a Spark-based solution providing probabilistic record linkage targeted to governmental datasets which was partially developed during this research. AtyImo was created as an alternative data linkage tool to support the integration of Brazilian governmental databases used in epidemiological studies. We have studied the current limitations of AtyImo to identify potential improvements when integrating our proposed methodology for exploring hybrid processors into the tool.

1.6.3 Step 3 – Parallel and heterogeneous modelling

This step was dedicated to the description of each algorithm developed during this research. Firstly, we describe the algorithms targeted to CPU execution and some parallel considerations useful to understand performance gains obtained in other processing units. After that, we present the algorithms targeted to GPU, multi-GPU and heterogeneous execution.

1.6.4 Step 4 – Performance and accuracy evaluation

One important step of this work was dedicated to providing some discussions about hardware limitations and performance gains considering different workload placements and parallel setups. We have also spent some time to the compararison between parallel (heterogeneous) execution and AtyImo (Spark-based) execution.

1.7 DATASETS

The real datasets used as proof of concept were SIM (Mortality Information System) for Brazilian states of Sergipe (SE), Santa Catarina (SC) and Rondônia (RO) and also a 5-year extraction of the CadastroÚnico (CADU) databases, totaling 100 million records). This dataset was used for assessing the accuracy of the proposed data linkage method. For the majority of scalability tests, we have used a synthetic dataset from different sizes described in each experiment. The datasets used in this study are detailed in Chapter 4.

1.8 METRICS AND EXPERIMENTAL SETUP

We intended to evaluate application performance in terms of runtime and accuracy. The parallel performance of the probabilistic comparison step is given by the speedup factor and execution time. Accuracy is assessed by recall, precision and positive predictive value (PPV) which refers to the number of true positives observed in the “match” set (i.e., the

number of record pairs correctly retrieved).

We have used three different configurations for testing our proposed methodology:

- **Heterogeneous environment 1 - Test Purpose:** Host Node: 4 Intel Xeon Quadcore CPU, 2.93 GHz, 24M of cache memory, 130GB of main memory. Device node: 2 Tesla k40 GPU with 28 Stream Multiprocessors (SM) and 64 Stream Processors (SP), CUDA version 6.0.
- **Heterogeneous environment 2:** Host Node: 2 Intel Xeon Gold 6148, 2.40 GHz with 20 cores each, 196GB of main memory. Device node: 2 Tesla P100-SXM2 with 16GB of main memory, 56 multiprocessors and 64 CUDA cores, CUDA version 10.0.
- **Heterogeneous environment 3:** Host Node: 2 Intel Xeon Gold 6148, 2.40 GHz with 20 cores each, 196GB of main memory. Device node: 2 Tesla V100-SXM2 with a total memory of 32GB and 5120 CUDA Cores, CUDA version 10.2.

1.9 CONTRIBUTIONS

This research significantly contributes to the state of the art by extending existing studies on functional performance analysis considering large workloads and the consequent impact of factors such as data transfer overhead/prediction to construct an efficient data partitioning model for environments comprised by heterogeneous processing units. Another important contribution is the application of this study to solve an open problem related to the poor performance of traditional tools in probabilistic record linkage applications.

Our contributions can be summarized as follows:

- We extended our previous approach (Spark-based pipeline) to provide an efficient parallel method to solve probabilistic record linkage that considers the computation power of other processing units (besides CPU) and to use a native programming language and parallel paradigms to make more efficient use of hardware.
- We developed a technique for splitting a traditional linkage pipeline at the comparison step and performing efficient workload division depending on existing processors and on the characteristics and volume of the input data.
- We have applied our solution on real government datasets in a project aiming at to integrate large-scale (100 million records) socioeconomic data with public health data split in disparate sources.
- The solution presented in this research has shown satisfactory recovery and classification metrics, reaching a minimum value of 97% and a maximum of 100% in terms of accuracy.

This chapter presents the updated references related to this work that was shared in the scientific community by the time of this thesis writing.

RELATED WORK

Current scientific researches that propose the investigation of high-performance techniques improvements to solve probabilistic big data linkage are still insufficient to deal with open problems imposed by recent big data specificities: i) scalability for large databases, maintenance of a high-quality result for linked pairs (due to the variety of big data sets), iii) ensuring privacy and confidentiality of the information being linked, and performance necessary to guarantee timely responses, indispensable for dealing with datasets in big data contexts.

Scalability, in general, is entrusted to big data frameworks that impose a specific programming model (such as key/value pairs) and employ a high level of transparency for development. On the other hand, high-performance implementations lack efficient treatment for in-memory workload requirements. Even though HDFS be the engine used by Hadoop, MapReduce, Spark, and others, It is also challenging rely on this distributed file system when using HPC clusters given the need for a large amount of local storage space (ISLAM et al., 2015).

However, we can find some works that seek to improve the traditional record linkage application's performance. We summarize the most relevant researches focusing on those that propose improvements in traditional RL implementations and also those that exploit the high-performance computing architecture to reduce the execution time of such applications.

Good scalability results are shown by (GSCHWIND et al., 2019) that employ a rule-based record linkage method over a real-world system. They also propose a native programming language and the load of the largest possible portion of the dataset in main memory to maximize performance. Although this work presents satisfactory performance analysis and results (linear scaling with the number of nodes added) for RL problem in multicore environments, heterogeneous processing units specificities were not considered.

An OpenCL based application was proposed by (RASCH et al., 2019) that considers modern high-performance architectures composed of multi-core CPU and many-core GPU

processing units. The authors achieved good speedups of up to 80 times. However, the solution does not demonstrate scalability measures for large workloads, using input data limited to 2^{20} (number of records). (SEHILI et al., 2015) and (FORCHHAMMER et al., 2013) also show OpenCL based implementation that has the advantage to unlink the solution to a given manufacturer, but lacks scalability and heterogeneous optimization considerations.

Considering that the cross-comparison required for the RL methods impose a quadratic growth in the execution time, some strategies are adopted to reduce the search space’s dimensionality. (KÖPCKE; RAHM, 2010) propose an approach to achieve better execution times, and different directions to accomplish this deed are presented in (CHRISTEN, 2011) and (MESTRE; PIRES; NASCIMENTO, 2015).

Despite presenting a study on the scalability of Record Linkage applications and a parallel implementation based on the distributed framework Apache Flink (CARBONE et al., 2015), the authors of (FRANKE et al., 2019) direct their efforts towards blocking strategies capable of relieving the quadratic complexity of the comparison process and not on the application’s capability to use an increasing number of processing units. (VAT-SALAN; CHRISTEN; RAHM, 2020) also point out scalability limitations of previous privacy-preserving Record Linkage methods, but its experimental evaluation considers only the comparison between runtime and memory size required to linkage. Although they propose an efficient approach to link large-scale datasets, the authors recognize the need to better explore distributed computing and parallel processing approaches.

In-home approaches have proposed different methodologies to circumvent the same problem when comparing all record pairs in a big data context. To reduce the search space for integrating two datasets, (BARBOSA et al., 2019) recently proposed the use of indexing methods based on predicates using the Lucene API (LUCENE, 2010). The reduction of the search space significantly optimizes the cost of computing in the identification of pairs. However, there is a direct relationship with the accuracy penalty that must be carefully considered, since the use of blocking increases the chances of false negatives (i.e. records with a great probability of being similar are placed in blocks that are never compared by the algorithm), especially in studies that require maximum exactness. Besides, timely responses with the guarantee of privacy preservation include new challenges in building an efficient and versatile integration tool. As blocking is more beneficial for reducing the execution time, the parallelization of computation becomes the main strategy to guarantee optimum performance without losing accuracy.

This chapter presents the theoretical foundation for the development and understanding of the developed research.

LITERATURE REVIEW

3.1 HETEROGENEOUS COMPUTING PLATFORMS

Nowadays, HPC ecosystems rely on the coordinated use of different PUs (processing units) to promote cost-effectiveness and accelerate applications that demand high computational capabilities. This new composition of high-performance computing scenarios is also the result of the scientific and industrial constant search for increasingly specialized hardware concerning the computational power reflected by the theoretical performance and number of floating-point operations per second.

According to (MAHESWARAN et al., 1999), heterogeneous computing can be understood as the coordinated use of different types of machines, networks, and interfaces to maximize the combined performance and efficiency of an application. In this work, we consider as heterogeneous computing the collaborative use of different processing units to speed up related tasks. Therefore, the goal is to deliver to each architecture the most appropriate tasks, considering the advantages and disadvantages of each hardware, as well as to prevent any available PU from remaining idle while parallel tasks are waiting in the execution queue. In addition to the performance gain, power efficiency is also an extremely important point when using hybrid environments. In most applications, the average utilization of a single GPU or CPU remains low. This is because after tasks start running on the GPU, the CPU remains idle waiting for result data, which means a large waste of computing power. Similarly, while the device waits for data that is taking too long to arrive (due to the low throughput of the interconnect channel), it remains idle.

Heterogeneous computing techniques seek to minimize these problems. Not surprisingly, heterogeneous computing environments represent ever-increasing numbers on the two lists of the highest performing and most efficient supercomputers: TOP500¹ and Green500², respectively. The success of this type of computing infrastructure depends on the ability to design, implement, and assign particular tasks to specific resources.

¹<https://www.top500.org>

²<https://www.top500.org/green500>

However, the biggest challenge to achieving efficient collaborative computing is related to the difficulty of surrounding the differences between the architectures that make up the heterogeneous supercomputing environment and the provision of a single layer of abstraction for different paradigms and specific programming models. Not surprisingly, unique optimizations for a particular processor are extremely inefficient when used in a hybrid environment. The fact is that, in terms of performance, power efficiency and development (programmability and portability), a great effort is needed to achieve an effective solution because heterogeneous environments are still difficult to manipulate. New heterogeneous computing techniques have been presented to exploit the potential of these systems and meet future performance goals in a *exascale* context (BERGMAN et al., 2008). In order to utilize the resources of such multiprocessing systems efficiently, parallel applications need to be re-engineered, as well as it is needed the development of new programming models, tools, and algorithms since hardware resources are usually built for a specific architecture. Besides, it is necessary to take into account the workload between different PUs (ZHONG; RYCHKOV; LASTOVETSKY, 2015) using data partitioning and load balancing methods.

3.1.1 Characterization of heterogeneous platforms

Multiprocessor systems have made High-Performance Computing (HPC) an increasingly enhanced reality. In practice, it is possible to add computational power according to the need, availability, and nature of the application. In today's HPC scenario, it is common to add manycore processing units, called accelerators, to distributed computing nodes composed of traditional multicore processors. Because they have a massively parallel architecture, such coprocessors are often used to improve the performance of algorithms with high independence between their tasks. Figure 3.1 demonstrates an example of a hybrid platform consisting of different interconnect links and multicore and manycore (GPU) processing units.

Ideally, the use of different multicore and manycore processing units should be transparent to developers by the use of automatic partitioning and load balancing methods capable of efficiently distribute parallel tasks over the heterogeneous hardware. Also, it would be appropriate for parallel application modeling to fit the architecture in which it is running, considering that each platform requires a unique resource management model.

While multicore processors are optimized for latency, manycore processors are optimized for bandwidth. In other words, one can take advantage of the fact that CPUs operate at a higher frequency and have a larger cache size to minimize latency issues. In this way, it is possible to allow small parallel tasks to be performed by the CPU in a low time. Although the number of cores in a GPU is much higher than in CPUs, GPU cores operate at a much lower frequency and their cache size is reduced as the main memory. Therefore, for a GPU compensate the time spent with data transfers from the host memory to its local memory, a good strategy is to send large tasks that can be processed using the SIMD (*Single Instruction Multiple Data*) paradigm. Through this approach it is possible to solve large scale problems that traditional computational solutions would not be able to do. In this context, a challenge is how to best distribute the application

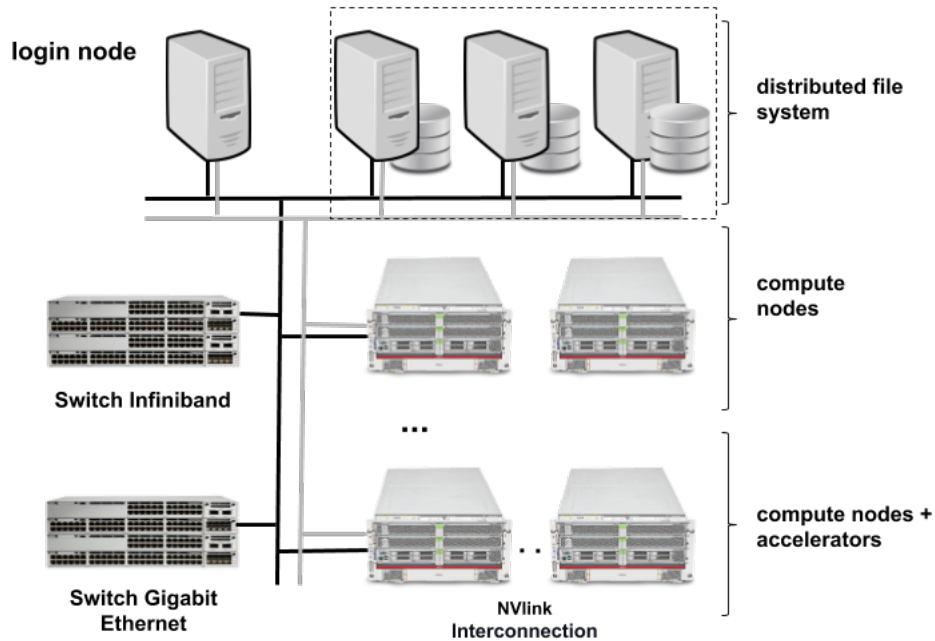


Figure 3.1 Hybrid architecture with different interconnection links and PUs (CPUs and GPUs).

tasks among the processing groups of cores ensuring better performance (considering the shared and distributed memory usage), as well as ensuring scalability and portability across multiple platforms and operating systems.

In the literature, infrastructure solutions that consider the shared use of CPUs and GPUs are commonly referred to as Collaborative Computing, Hybrid/Heterogeneous Computing, or Co-Processed Computing (MITTAL; VETTER, 2015). In general, the idea prevails that since GPUs are coupled to traditional multicore nodes, and considering that most general purpose applications are still primarily optimized for multicore execution, such boards are the units classified as "accelerators". In this work, the term PU (Processing Unit) is used to generalize both CPU and GPU.

To clarify the evolution of heterogeneous multiprocessor systems, it is important to consider the evolution of parallel hardware concerning the number of transistors, the number of cores and memory hierarchy.

- Number of Transistors: Until recently, the number of transistors on a commercial CPU processing unit integrated circuit was around 1 billion (WENDEL et al., 2010). In 2019, AMD's Zen 2 Epyc Processor (SUGGS; SUBRAMONY; BOUVIER, 2020) reached the mark of 39.54 billion transistors, being 8.34 billion transistors on a single die. In accelerated processing units, this account reaches much larger values: 54 billion transistors in 2020 Nvidia's A100 (KRASHINSKY et al.,) and 43,3 billion transistors in 2019 Intel's Stratix 10 GX 10M FPGA (INTEL, 2019).
- Number of cores: Initially, CPUs contained only one or two computation cores. In

2019, the AMD Zen 2 had, in turn, 64 cores (128 threads). Compared to NVIDIA's Ampere-based A100 accelerator, the number of CUDA cores reached 6912.

- **Memory Hierarchy:** As the number of cores increased, the size of the Last Level Cache (LLC) has also significantly increased. Currently, it is possible to find microchips with 144 MB of L3 cache, like the AMD Ryzen Threadripper 3970 (AMD, 2019) processor. As for GPUs, the Nvidia Tesla V100 graphics card with 32 GB of memory and 6144 KB of cache size is popular in current production environments and the Nvidia Ampere-based A100 graphic card with 40 GB of memory must become popular in the future as well.
- **Bandwidth:** In 2018, the TOP500's fastest supercomputer was composed of a heterogeneous NVlink high-speed link architecture not only between graphics cards but also between CPU and GPU PUs. In practice, the NVlink connection between two Tesla P100 cards, for example, provides a transfer rate of approximately 40 GB/s, while for cards connected by a PCI-standard, this rate is around 10 GB/s. Besides, memory bandwidth for the latest Nvidia GPU until this day A100 is around 1555GB/sec.

Given their highly parallel architecture, graphics processing units can use thousands of threads to process a job in parallel, speeding up a job that would run up to 64 threads on a modern CPU. Also, for applications appropriately built for benefiting from this kind of architecture, GPUs are often more power-efficient when considering the Watt/instruction ratio. Seeking performance optimization is a critical and challenging task, especially when the same workload is launched to processors with different speeds and memory configurations.

Initially, to add computational power to a high-performance node using GPUs, it was common to connect different devices to the CPU through a PCI-e bus/switch, where it was possible to get around 533 MB/s. As workloads become larger (especially Artificial Intelligence training data) and GPUs become more robust, performance gains will only be proportional if interconnection technologies keep up with this evolution.

Currently, the bandwidth provided by the PCI-e bus is known to be a bottleneck for the performance of GPU applications. Especially for applications that perform several small-data transfers between CPU and GPU in an iteratively way (GREGG; HAZELWOOD, 2011). However, to compensate for this loss, a NVlink high-speed interconnect offering up to 12 times faster bandwidth than PCI-e connections has been proposed. This interconnection links both GPU and CPU processing units and GPUs arranged on the same computational node.

Table 3.1 presents a summary considering general characteristics of PUs and architectures. For each row in the table we consider which PU is most suitable in terms of the following metrics: a) Affordability defines the lowest current price at which this kind of processing unit can be bought or sold; b) Small files performance defines the PU that best handles data parallelism for small files; c) Throughput optimization defines the PU that optimized for delivery computation for a high number of instruction simultaneously; and

| Relevant characteristics | PU multicore | PU manycore |
|--------------------------|--------------|-------------|
| Affordability | x | |
| Small files performance | x | |
| Throughput optimization | | x |
| Latency optimization | x | |

Table 3.1 Advantages and disadvantages of a hybrid parallel application running on a heterogeneous platform composed of multicore and manycore PUs.

d) latency defines the amount of time for a PU access instruction and data and deliver appropriate answers.

3.2 ALGORITHMIC ISSUES CONCERNING HETEROGENEOUS COMPUTING PLATFORMS

One of the biggest challenges in building parallel and distributed solutions for big data scenarios is the computational infrastructure dedicated to data-intensive computing. Over the years, it has been possible to observe a constant evolution in the processing capability of computer systems. Until recently, Moore’s Law (SCHALLER, 1997) matched the reality, periodically showing faster and increasingly dense processors. However, in the current state of the art related to building integrated circuits, there are several obstacles to miniaturization. On the other hand, the scientific and industrial community is embracing other alternatives such as specialized architectures designed specifically for one sort of algorithms. Today, computer architectures have evolved to support parallel applications through manycore and multicore PUs.

In the big data era, HPC has evolved to deal with a different scenario since there is a significant challenge to manage, store and operate on the data to perform large scale data analysis (ISLAM et al., 2015). Despite this, the requirement for disk writing and reading (I/O) remains a bottleneck: their access speed and the impact caused by their controllers continue to negatively impact overall applications performance. That is the main disadvantage in using Hadoop for processing in-memory workloads interactively since *Reduce* tasks are I/O-bound when processing many large output files. The need for real-time responses to support decision making in the analysis steps has forced developers to find effective alternatives. Therefore, it is desirable that while processing and analysis are being performed, data (or at least most of it) remains stored in main memory and less I/O operations be performed. This strategy is adopted by current and widely used big data processing frameworks such as Spark (ZAHARIA et al., 2010). Opportunely, the cost of main memory resources has declined significantly, while the storage capacity has increased, allowing for the popularization of *in-memory* big data processing and analysis systems.

3.2.1 Workload-partitioning techniques over heterogeneous platforms

Considering the specific characteristics of multicore and manycore hardware presented in subsection 3.1.1, it is clear that there are more appropriate parallel problems for

one architecture than another. Due to these factors, different load distribution settings between CPU and GPU can lead to very different performance.

The chosen model for partitioning and load distribution will largely determine whether the overall performance of the heterogeneous application will be satisfactory. Data decomposition approaches consider, at a first level, how to partition data into heterogeneous processors to avoid load unbalances. This imbalance is common when an improper distribution is done and the processes involved in computing finish their work at very different times. Static load balancing strategies are useful when data location is important (data cannot shift between heterogeneous PUs) or when application performance for specific workloads is well defined. A major advantage of static approaches is that they incur less communication overhead between processes in a distributed memory environment as fewer messages need to be sent.

In practical terms, small problems are not suitable for running on GPUs. That is because the small blocks that will be processed in parallel do not utilize the full parallel potential of the hardware. Besides, small problems can better exploit the memory hierarchy locality in CPU environments, optimize cache usage and help to increase process throughput and thereby, improve response time. Parallel problems of coarse granularity can also be a problem to scalability in manycore environments, as they are made up of fewer blocks and more serialization for each execution flow. Tasks with non-uniform memory access pattern also need to be carefully allocated to avoid communication overhead and data search from main memory (high latency).

To achieve satisfactory performance, it is necessary to balance the load between the PUs involved in the heterogeneous environment. For this, it is of fundamental importance to consider four points: retention of shared resources, limitation of available GPU memory, the bandwidth of communication channels (both *host-to-device/device-to-host* and *device-to-device*) and the impact of non-uniform memory access (NUMA).

The application investigated in this work is parallel and data-intensive, consisting of in-memory workloads. This means that the expected workload is proportional to the size of the input data. Also, it can be expected that there is a fixed amount of computation that will be distributed among different threads.

Partitioning models propose load balancing based on the theoretical performance characteristics of processors. In this research, we consider constant performance measurement models based on historical execution time. When considering these types of models, it is necessary to start from two important assumptions. The first ensures that *the absolute speed of PUs does not depend on the size of the task to be performed, but can be represented by a constant value*. The second ensures that *the processing elements are independent of each other and therefore their speeds and performance can be measured independently*. Functional performance models were considered as they generate a characterization of the application workload and also a characterization of the hardware that makes up the target system. The objective is to model the performance of the specified application considering a certain amount of parallel threads and their allocation.

To achieve a better throughput rate – the number of tasks a processor can handle in a given time, which can be measured by the amount of data transferred from one node to another – it is common to use *batching* and *streaming* (PINNECKE; BRONESKE;

SAAKE, 2015). In practice, these strategies alleviate the problem of having to store the entire workload in memory at one time, as these loads are variable in size and PUs have limited main memory capacity (especially GPU accelerators).

Column-driven rather than row-oriented data transfers also contribute to reducing the overhead of communication channels compared to whole-table oriented transfers. However, in modeling real problems, row-oriented data structures (such as multi-dimensional arrays) are more common due to issues related to access speed, space optimization, and representation. To explore the partitioning of data to be transferred to the GPU, for example, it is necessary to alternate the shape of these structures concerning the orientation of their representation. Considering the memory limitation of graphics cards, the window size used for data transfers should not be too large. Therefore, the optimal window size should be set according to the memory capacity of the target device.

While Constant Performance Models (CPM) assume that the relative speed of processors does not depend on the size of the tasks handled, Functional Performance Models (FPM) models assume that such speeds may vary due to the processor and memory heterogeneity. In these scenarios, the speed of a PU may depend on the load assigned to other PUs due to resource contention. The speed of each processor, then, is represented as a function of the problem size. Speed is defined as the number of calculation units performed by the processor in a given unit of time. (CLARKE; LASTOVETSKY; RYCHKOV, 2011), (CLARKE et al., 2013).

3.3 BIG DATA PROCESSING AND ANALYSIS

It is possible to state that the big data phenomenon has changed the way how personal and corporate data are kept, managed, analyzed and valued, producing what we know today as the fourth industrial revolution. This context brought the recognition that understanding a dataset means to extract insights and answer questions. By 2020, there will be about 40 trillion gigabytes of data, i.e. 40 zettabytes and, despite statistics and forecasts, it is difficult to measure how fast data is growing. Throughout this process, the technology and data science community have dedicated a comprehensive effort to provide solutions that are compatible with the growing demand not only in terms of data volume but also related to the complexity of the analytical algorithms. Still, as illustrated by Figure 3.2, our ability to analyze this data grows at a much slower rate than it is required to keep up with data demand. Table 3.3 shows the evolution of big data processing and analysis technologies, highlighting examples of some products launched for this purpose.

Build fast data integration and analytical processes is particularly important due to two main reasons. The first one is related to the speed that data is generated. In many scenarios, data collection is continuous and the processing and analysis steps need to deliver results at the same pace, with the same quality, accuracy, performance, and efficiency, without incurring major delays. Secondly, it is important to note that the purpose of such integration is to provide a timely response compatible with decision making. This means that the person submitting a query needs to get the answers as soon as possible to ensure his applications moves forward.

The application of big data brought significant changes to the industry, making pro-

| | |
|------|--|
| 1997 | The problem of Big Data was defined by NASA. (COX; ELLSWORTH, 1997) |
| 1998 | Google was founded. |
| 1999 | Apache Software Foundation was established. |
| 2001 | The 3Vs were defined by Doug Laney. (LANEY, 2001) |
| 2003 | The Google File System was presented. (GHEMAWAT; GOBIOFF; LEUNG, 2003) |
| 2004 | Development of Google’s Big Table. |
| 2004 | MapReduce was presented. (DEAN; GHEMAWAT, 2008) |
| 2006 | Hadoop was presented. |
| 2006 | Yahoo developed Apache Pig on Hadoop. |
| 2007 | MongoDB was presented. |
| 2008 | Apache Hive, HBase and Cassandra was presented. (HIVE, 2013) |
| 2010 | Apache Spark was presented. (ZAHARIA et al., 2010) |
| 2020 | Exascale era. (ALOWAYYED et al., 2017) |

Table 3.2 Qualitative difference between volume, velocity and variety of generated data and ability to analyze it over time (Adapted from (KAY; HARMELEN, 2014)).

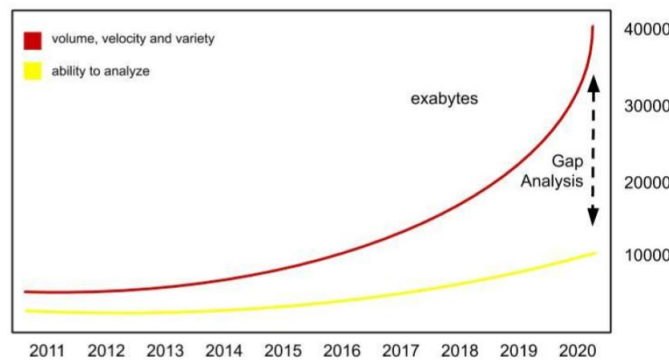


Figure 3.2 Gap between ideal and actual ability to analyze big data (LEMIEUX; GORMLY; ROWLEDGE, 2014).



Figure 3.3 Generic big data pipeline and knowledge discovery process.

duction processes more efficient, reducing costs, increasing production and allowing better efficiency in the use of expensive resources such as electricity. In administrative and management fields such as the case study presented in this research, the use of efficient big data technologies enable fraud detection/reduction, better employment of public money and evaluation of the impact of previous investments. In this context, data analysis refers to the application of some kind of data transformation intending to discover and extract knowledge (AMARAL, 2016). The analysis provided by the record integration process is

classified as "explicit" and has a low complexity if compared to the "implicit" analysis provided by prediction and inference methods that often use machine learning techniques to achieve a goal. A common process to all analysis methods, however, is the exploratory data analysis (initially proposed by (TUKEY, 1970)) which seeks to quantitatively or qualitatively extract the main characteristics of the data being analyzed.

Another important feature of big data analytics is the challenge posed by the cleanup and standardization step. Due to its immense volume and origin (various heterogeneous sources), it is always assumed that all data in this context is subject to inconsistencies, noise, and missingness. Poor data quality will significantly impact the accuracy of any integration process even if the developed algorithms have maximum accuracy. Preprocessing techniques, therefore, are part of a generic big data pipeline presented by Figure 3.3 and that can be explained by the following list:

- **Data collection and access:** It is the online or offline process dedicated to performing the systematic collection of data produced by multiple heterogeneous sources. Data storage devices and architectures currently used to keep and access these data also have changed after the big data era. It is no longer possible to rely on traditional filesystems since I/O speed is a limiting factor. The information must be accessed easily and promptly. Therefore, distributed filesystems are a proper option and should be considered.
- **Preprocessing:** Common operations for data harmonization include string formatting, standardization of fields, removal of special characters, insertion of specific values for missing data, removing stop words etc. This step is particularly important to improve the input data quality and reduce the negative consequences that noise may cause to this process. After preprocessing, data can be handled by data mining functions and stored for future analysis.
- **Transformation:** This process is related to all kinds of alteration of an event from one form to another including translation (it performs a single event in, single event out operation), splitting (takes a single incoming event and emits a stream of multiple events), aggregation (takes a stream of incoming events and produces one output event that is a function of the incoming events), composition (takes two streams of incoming events and operates on them to produce a stream of output events) etc.
- **Processing:** This procedure concern the data mining process, aimed at to analyse data using big data analytics techniques.
- **Evaluation:** This step includes procedures to analyze how accurate and precise are the results generated at each step. It must be used to provide a metric to validate the quality of the whole pipeline.

3.4 DATA INTEGRATION IN THE CONTEXT OF BIG DATA

Turning raw data into useful information that can add value and support decision making is an increasingly urgent requirement. The problem of correlating records from different

dataset domains representing the same real-world entity is known as *data linkage* or *record linkage*. This problem is a critical step in the integration process, especially when considering the huge volume of associated data. In addition to the volume, another major challenge is the lack of a unique identification key able to ensure a deterministic linkage between the different datasets. Therefore, it is necessary to have a set of attributes through which a matching probability can be determined. This method is called probabilistic record linkage and demands a planning step to reasoning about which linkage keys should be used to decide about *match* or *non-match*.

The probabilistic linkage model is a technique widely used in various industrial, commercial and scientific scenarios. The literature presents several methods and tools capable of adapting traditional *join* operations and cartesian products to the specificities of big data integration applications. Generally, these problems have a high complexity regarding I/O, memory consumption and runtime.

Government, finance, and health have experimented with various methodologies for employing more efficient data integration strategies along these lines. Specifically, in the healthcare domain, efficient data integration allows integrated management and monitoring of outpatient records, as well as longitudinal follow-up of the epidemiological and socioeconomic history of the population, generating indicators of significant importance concerning the determinants of an outcome (a disease of interest, for example).

There are two main strategies for performing data correlation: the most flexible to design and implement is deterministic, which links two or more records using one or more exact match keys. When it is not possible to rely on common linkage keys, the strategy used is to compare secondary identifier attributes through probabilistic methods. Figure 3.4 describes the attributes of two records A and B and indicates that, ideally, values for which there are only two possible responses (equal or not equal), such as date of birth and county code, are more suitable for exact comparison using the deterministic method. For values that allow an approximate result (ranging from 0 to 100% of similarity), such as names, probabilistic methods are more suitable.

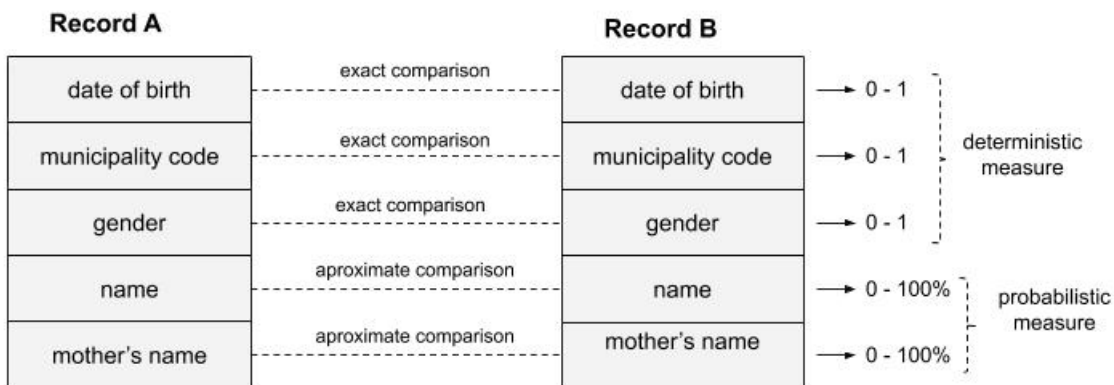


Figure 3.4 Pairwise comparison of two records (A and B) containing deterministic (exact) and probabilistic (approximate) attributes.

The main challenge for probabilistic data integration is the true-pair hit ratio since

there is a high chance of duplicates between the attributes used in the comparison and the possibility of divergences in the representations of the same record in different contexts. Besides, in order to obtain maximum accuracy, the probabilistic method needs to perform a large number of comparisons (quadratic order) and perform complex similarity calculations between the compared records. These two factors make probabilistic linkage methods computationally demanding, resulting in high execution times for generating a resulting output containing the desired true pairs. To better understand the challenges involved in this process, we formalize the main problems of probabilistic record linkage in the 3.5 section.

3.5 FORMULATION OF THE RECORD LINKAGE PROBLEM

The theoretical basis underlying the probabilistic record linkage method was first presented by (FELLEGI; SUNTER, 1969) and its guidelines are still widespread. Let A and B be two separate datasets and let R_a and R_b be the set of all records of A and B , respectively. The probabilistic record linkage method aims to classify all pairs (a, b) from the product $A \times B$ into two distinct sets: M being the set of positive pairs (matches) and U being the set of non-matches. It is also possible to classify pairs as “doubtful” (false positives and false negatives) into a third set. The expected cardinality for the linkage operation is 1 : 1 (assuming no duplicates will be included), which means that the set of linked pairs will have the size of the small dataset at most.

Considering that pairwise comparison is performed using identifier attributes common to both datasets (such as name, residence information, date of birth and parents name) it is necessary to establish a similarity index to decide whether or not a pair represents the same real-world entity. In probabilistic linkage, the similarity index represents how permissive the method will be when adding pairs to the true pair group (M). In deterministic linkage, the associated similarity index is a boolean value, since there are only two link possibilities: *match*: 1 and *non-match*: 0. Deterministic linkage can be used for all types of attributes. However, to increase the reliability of the method, it is recommended to use it only for those values of short variation and less susceptible to errors, such as gender, race/color, binary fill values (yes/no) etc.

3.5.1 Privacy preservation in probabilistic record linkage

Preserving the privacy of nominal and identified data is a fundamental requirement in many contexts. Computer systems are therefore committed to implementing solutions that ensure efficient and secure data management. The implementation of data anonymization routines should be strong enough to make it impossible to re-identify the original data by using irreversible transformations of plain text data into code. In this way, data that was previously confidential and followed strict curation criteria can be shared with other parties.

These anonymization and de-identification methods are common, for example, in epidemiological, hospital and outpatient studies in various health sectors. In applying probabilistic linking procedures for data integration, the most relevant data are precisely those

considered as identifiers. Thus, the implementation of de-identification methods is not applicable, since such methods require the removal of the attributes of interest, called quasi-identifiers. The anonymized data is expected to be sufficiently representative, i.e. the similarities and differences between the records are represented in the same proportion. For this, coding methods should be developed aiming at to generate anonymized records with maximum accuracy (plain text data representation). The record correlation process, in this case, is known as *privacy-preserving record linkage* and imposes new requirements on the traditional correlation methods (HALL; FIENBERG, 2010).

Another factor that should be considered in the study and development of anonymization approaches concerns the computational performance required for record generation and manipulation. The first challenge is to build the transformation routines. In this sense, it is necessary to consider the cost for decomposing the plain text and the algorithms that map the text that has been decomposed, in fixed-length code. These procedures involve conventional one-way encryption, for example, and *hash* functions are a common strategy in composing such algorithms. However, the reliability of anonymization methods varies for each implementation.

Bloom Filters (BLOOM, 1970) appears in this context as a viable alternative, not only to protect personal information and ensure data privacy but also to improve comparison performance, since comparing two *strings* is a high costly operation. The method guarantees a satisfactory result regarding the execution time and storage space of the records to be compared. Bloom filters are effective in maintaining the correct distance between represented strings while solving privacy preserving issues.

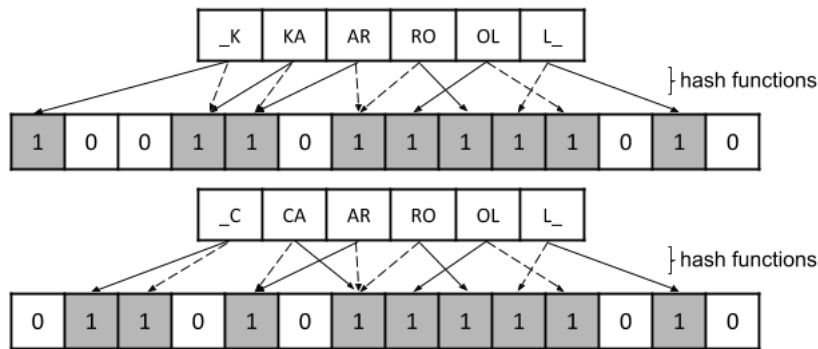


Figure 3.5 Example of Bloom filters: applying hash functions to bigrams of a string.

The approach is to represent any record using an encoded data structure, represented in a vector *bitmap* V of size S . At first, all elements of the vector are zeroed. The string to be transformed must, therefore, be decomposed into bigrams (consecutive character pairs, including blanks). The decomposition into bigrams comes from studies concerning the natural language process. Each bigram is then submitted to H encryption (hash) functions (MD5 and SHA1, for example) that will generate a summary. The process of constructing a string transformation key in a Bloom filter code should be kept isolated from the application to increase security and prevent attacks. Figure 3.5 demonstrates the process of transforming a flat string into a bit vector.

An entry \mathbf{E} is submitted to a function \mathbf{Q} , generating an encoded record. So if $\mathbf{Q}(E_1) = V_1$ and $\mathbf{Q}(E_2) = V_2$, and $V_1 \equiv V_2$, it is considered V_2 , and V_1 are positive pairs. The equivalence between two records in Bloom filters will be given by a similarity function that will calculate how similar the two records are. This numerical result ranges from 0 to 1 and represents the percentage of agreement between two records. The Bloom filter method does not allow false negatives and false positives, in turn, can be reduced by setting an appropriate length for the vectors.

3.5.2 Indexing and blocking construction

Historically, much has been investigated and developed to reduce the dimensionality search space for candidate peers and reduce the total execution time for linkage methods applied to large datasets. This method, also known as indexing (ELFEKY; VERYKIOS; ELMAGARMID, 2002), applies some criteria to increase the chance that true pairs will be compared first, to increase accuracy. The most widespread strategy used for this purpose is to group or sort records into blocks according to some similarity rule. Several strategies for blocking and indexing are proposed by scientific community (CHRISTEN, 2011).

The justification for the use of this strategy is because the vast majority of comparisons made between candidate pairs to establish a single equivalence are unnecessary. Since the attributes differ sufficiently to exclude that pair from the set of positive matches (*matches*), excluding them from the search space represents a significant gain in overall performance. The standard approach to building blocking functions is based on a subset of common attributes to both datasets used as blocking keys (or predicates). Figure 3.6 demonstrates the predicate-based blocking strategy. This approach ensures that errors in one of the clauses do not prevent that record from being correctly grouped into the same group as its true pair. Pairs removed from the same block are automatically classified as non-match. That is a reasonable fact since it is known that the vast majority of comparisons (without blocking) would be performed between records that do not match. Pairs that belong to the same group are then, compared, following the similarity measure function established and classification rules previously defined.

The disadvantage of using blocking methods is that if a true pair is mistakenly included in separate blocks, they will never be compared, even if the similarity calculation is sensitive enough to identify them as a true pair. This can occur when a pair cannot be represented by the chosen blocking keys. A simple example is to apply excluding attributes such as *gender* (in case of identifying people) in the composition of the blocking key. If there is an imputation error in filling that individual's gender, it will be automatically excluded from the comparison. To mitigate this issue, one can implement composite blocking keys, such as predicate blocking, for example. However, the only way to ensure that all true pairs will be compared is by comparing all the records that make up the domain.

Thus, whenever it is possible to apply similarity comparisons considering all available records in the search space instead of applying blocking strategies, this is the alternative that will bring better accuracy results. The two main factors that make this approach

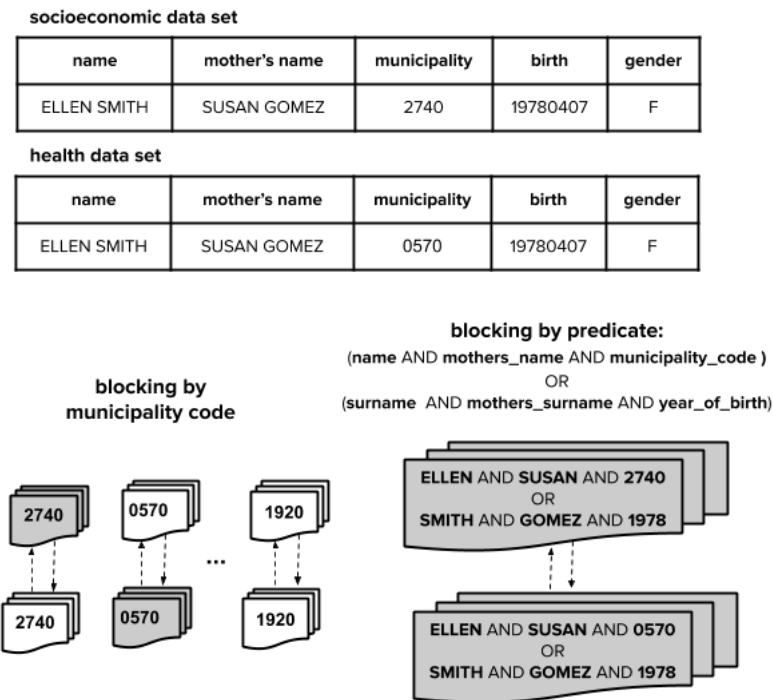


Figure 3.6 Example of a blocking predicate implemented in this work.

unfeasible are domains with a high number of records and insufficient hardware architectures. A hardware infrastructure and software solution capable of executing massively parallel algorithms is therefore not a recommendation but a crucial requirement for a wide range of applications.

3.5.3 Similarity calculation

To quantify the similarity between two-string symbols, several measurement approaches have been proposed in the literature. Most similarity calculating methods are based on edit distance, i.e. the number of editing operations that are required to transform one string into another, the length of the longest common string, and the number of n-grams (common substrings). The edit distance method (RISTAD; YIANILOS, 1998), also known as the Levenshtein distance method, is effective for short strings (where few differences between strings are expected). Regarding binary similarity measures, many approaches can be found in the literature. Such approaches consider definitions expressed by matching values between two strings to be compared in a binary fashion.

The Sørensen-Dice (DUARTE; SANTOS; MELO, 1999) index guarantees equal outputs for equal vectors and ensures a more generic representation for the similarity between common strings. When applied to binary data, such as the bit vector generated by the transformation using Bloom filters (3.5.1), this function returns a floating value between 0 and 1. Whereas N_{bg_a} is the number of bigrams of a string that has been mapped to 1 in a given binary vector a . Assuming that N_{bg_b} is the number of bigrams of a string

that has been mapped to 1's in a given binary vector b . The Sørensen-Dice similarity index is given by the equation 3.1.

$$SD = \frac{2|N_{bg_a} \cap N_{bg_b}|}{N_{bg_a} + N_{bg_b}}, \quad (3.1)$$

In practice, the Dice Index is a more sensitive measure in heterogeneous data (MC-CUNE; GRACE; URBAN, 2002). However, the accuracy and representativeness of the method largely depends on the number of sub-clusters produced to compose the vectors to be compared. This number must not be so small to harm the representativeness of the original *string*, nor so large to harm performance, as it is expected that a large number of calculation and comparison operations will be performed over this data.

This chapter describes the case study applied in this research which corresponds to the probabilistic integration of records from large government databases. We discuss the main challenges and open issues in this context to link such databases with maximum accuracy.

CASE STUDY: INTEGRATING LARGE GOVERNMENTAL DATABASES

One domain where probabilistic record linkage is frequently applied is Public Health. In impact assessment studies, for example, it is necessary to use individual search methods in the analysis group to prove hypotheses, observe events, and discover unknown patterns. Similarity functions applied to records should circumvent minor variations from possible imputation errors, abbreviations, and different conventions. The goal is to obtain a method that can minimize false positives and false negatives. Analyzing a large amount of data is not a trivial task. Extraction, transformation, storage, processing, retrieval, and distribution techniques need to be increasingly accurate while ensuring satisfactory performance. In Subsection 4.3 we demonstrate the problem of linking Brazilian government databases, the main challenges encountered and open problems. The pipeline and data flow description of our proposed solution is presented in Section 5.

4.1 LONGITUDINAL STUDIES BASED ON POPULATION COHORTS

Being classified as longitudinal analysis, cohort studies stand out as a commonly applied method in Epidemiology to follow-up individuals and explore evidence-based answers. In such studies, individuals are classified following some exposure status and then they are tracked over a period of time (hours, days, years) to confirm the incidence of a given observation.

According to (OLIVEIRA; PARENTE, 2010), the specific advantages of a cohort study are i) the possibility of distinguishing the temporal relationships between exposure and outcome, since the former precedes the latter; ii) the possibility of assessing multiple outcomes; and iii) the possibility of assessing multiple exposures. In this context, the cohort used as the basis of our case study focuses on the incidence of a given epidemiological occurrence (such as a disease). Through the incidence results observed for both

groups (exposed and non-exposed), it is possible to identify which exposures precede the outcome and which causal relationships can be established.

Such studies generally use longitudinal datasets to track individuals over a period of time. The longer the time period and the greater the number of records covered by the study, the greater the complexity in building the unified dataset. Observational studies applied to health seek to answer details about when, where, and how a disease or outcome occurs and also to observe how the occurrence of a certain disease varies according to the existence of an event of interest, using primary care (clinic) and secondary care (hospital) data. This association as a way of establishing a causal relationship between the occurrence of the disease and the event of interest commonly requires the use of integration methods between primary and secondary data.

Some Brazilian studies, which follow the observational method for relating administrative datasets containing socioeconomic data with health indicators, show that the use of the Bolsa Família (PBF) – one of the largest conditional cash transfer programmes in the world, which provides monthly payments for poor families – is associated with a reduction in homicides, hospitalizations for violence and mortality for infectious diseases. The study presented by (MACHADO et al., 2018) shows that for each percentage increase in the uptake of the Bolsa Família payment, the homicide rate decreased by 0.3% and hospitalizations from violence by 0.4%. (CARTER et al., 2019) considers the social protection of PBF as a non-negotiable component to reach the tuberculosis elimination.

(LIVINGSTONE et al., 2015)(HAWKINS et al., 2016)(HOLMAN et al., 2008) highlighted the value of using cohorts built from the information combined from multiple domains, to allow significant knowledge acquisition from the studied dataset. The case study of this research aggregates health information with socioeconomic data from the Brazilian population. The importance of integrating this information, in addition to statistically improving the quality of the study, is essential to establish conclusions that guide evidence-based decisions, especially in contexts where the organization, maintenance and availability of data is precarious.

4.2 BRAZILLIAN GOVERNMENTAL DATABASES

Different from most developed countries, Brazil is characterized by its high number of people (around 209 million inhabitants according to the 2010 Census), as well as by the large social inequality of its population. Over the past 20 years, structural (social and economic) changes have also reflected in a considerable improvement in the health of the population (RASELLA et al., 2013). This sort of research requires the longitudinal study of existing information in several different information systems. Applied to epidemiological studies, cohort studies seek to track specific health occurrences in a large group of people. In Brazil, a major challenge was building a cohort of 114 million individuals in a state of poverty or extreme poverty who have received payments from PBF between 2006 and 2015¹. The objective of this effort was to understand the impact of cash transfers on health outcomes and to assess the effectiveness of public policies, as well as provide evidence for future decision making about the destination of public resources.

¹<https://cidacs.bahia.fiocruz.br/en/platform/cohort-of-100-million-brazilians/>

| Information System | Coverage | Description |
|-------------------------------|--------------|---|
| Cadastro Único (CadUnico) | 2007 to 2015 | National registrar for social benefits that manages in a centralized way socioeconomic information of the population in poverty or extreme poverty or able to benefit from more than 20 social and protection programmes. |
| Bolsa Família (PBF) | 2007 to 2015 | Conditional cash transfer programme serving families in poverty or extreme poverty identified in CADU. |
| Hospitalizations (SIH) | 1998 to 2011 | State-level coverage of hospitalization. information on individuals admitted to public or private hospitals pertaining to the SUS network. |
| Mortality (SIM) | 2000 to 2012 | National coverage of Brazil's mortality data. It includes time of death, type of death, cause of death, date of death, date of birth, gender, race, education, pregnancy, and others. |
| Disease Notifications (SINAN) | 2000 to 2010 | It is filled with episodes on diseases of compulsory notification (a list of 52 different diseases). In this research, we have used data on Tuberculosis and Leprosy. |
| Live births (SINASC) | 2001 to 2012 | Epidemiological information regarding live births reported throughout the country. This database includes information on individuals at time of birth, including maternal information. |

Table 4.1 Government databases used in this research.

To achieve this goal, the Cadastro Único (CadUnico) database, which corresponds to a central registrar storing socioeconomic information (demographics, family composition, education, housing characteristics, sanitation, among others) of individuals eligible for several Brazilian social programmes, was used as a starting point (baseline).

By 2015, 114 million Brazilians in about 40 million families were registered in CadUnico, which is equivalent to approximately half of the entire Brazilian population. To build a single longitudinal dataset, a year-by-year merge was made from 2003 to 2015, tracking temporal modifications of the same record (duplication from exiting identification keys, as well as re-entries) and data harmonization between different versions and schemes. It was considered as inclusion criteria the date of the first registration of each individual. The construction of this cohort, therefore, took into account the full range of Brazilian municipalities representing a high-level of population coverage. Bolsa Família records are stored in monthly payrolls for the period from 2004 to 2015 and are linked to CadUnico in a deterministic way through the citizen's Social Identification Number (NIS). There are several conditions for an individual to be eligible for PBF payments, including school attendance of children and adolescents, as well as vaccinations and nutritional surveillance².

²<http://mds.gov.br/aceso-a-informacao/mds-pra-voce/carta-de-servicos/gestor/bolsa-familia/condicionalidades>

The integration of health datasets with the socioeconomic information of CadUnico allows not only to assess the impact of assistance and conditional cash transfer programmes but also to identify the social determinants for each outcome. To provide a tool able to handle such diversity of information, we had access to datasets from the Brazilian Public Health System (SUS), comprising data on mortality, live births, compulsory notifications of certain diseases, and hospitalizations. These datasets were provided by the Institute of Collective Health of the Federal University of Bahia and later by CIDACS - Center for Data Integration for Health (reference) that also offered statistical and epidemiological support to validate the results. During the development of the first stage of this research, we used datasets listed in Table 4.2 to identify the data patterns for each specific domain and map the transformations necessary and adjust the parameters of the probabilistic comparison to ensure the best accuracy values for epidemiological research.

An important feature that makes the integration of these datasets an extremely complex process is the variation in data quality. While mortality datasets have high quality, other datasets, such as hospitalization, have unsatisfactory quality for many states and specific health units. Among socioeconomic information, it can be expected a high rate of missing values for groups such as the homeless and children, inconsistent coding patterns, different schemes for datasets from independent domains. Thus, a strict preprocessing criterion must be applied to all domains involved in the integration. Besides, the absence of a unique identifier that could provide a deterministic linkage between health and administrative datasets is also expected. These challenges must be taken into account when selecting common attributes to probabilistically integrate these datasets.

4.3 THE 100 MILLION BRAZILIAN COHORT

The linkage involving the presented datasets was intended to collaborate with the 100 Million Brazilians Cohort (CIDACS, 2015), a joint Brazil-UK cooperation started in 2013 intending to build a population cohort to be used in several types of epidemiological surveillance studies. The linkage between these databases allows for obtaining data that relate to various outcomes and specific occurrences with the social determinants associated to them.

The linkage between CadUnico and PBF can be made deterministically, as there is an obligatory identifier key in both datasets – the Social Identification Number (NIS). However, for linking CadUnico or PBF to health care databases (from SUS) we should apply probabilistic methods, considering that there is no common identifying key.

Due to technological limitations in managing a cohort that encompassed the entire CadUnico cohort, as well as given the bureaucracy of gaining access to datasets since the promulgation of the Information Access Law ³ and new data privacy protection rules, few research studies have employed linkage methods to evaluate how governmental policies targeting the social determinants of health can affect health outcomes (ALMEIDA et al., 2019).

The business rules for all stages of linking (including cleansing, standardisation, harmonisation, blocking, selection of cut-off points, and data retrieval) depend on the domain

³http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2011/lei/l12527.htm

to which researchers belong to, typically epidemiologists and statisticians.

This chapter describes the development of the multiplatform tool AtyImo-H that implements a probabilistic data linkage pipeline for structured datasets. We discuss the overview of our solution, the data integration model, and the pipeline stages implemented.

ATYIMO-H DATA LINKAGE TOOL

5.1 ATYIMO-H

AtyImo is a pipeline-based tool that provides linkage methods for massive datasets. As stated in the Chapter 4, it was initially developed to support the linkage of socioeconomic data and public epidemiological data (health care) to build the 100 Million Cohort (BARR-RETO et al., 2017). The granularity of the integration is focused on the individual's data – candidates or beneficiaries of social programmes and their medical episodes (hospitalisation, diagnosis of a disease of compulsory notification, live births, and deaths).

This kind of data integration has a significant requirement for high performance due to the huge volume and diversity/heterogeneity of datasets domains and schemes that demand complex routines to perform similitiry calculations. Besides, to deliver a result with maximum accuracy and avoid false negatives as much as possible, several procedures are performed in a redundant manner: two verification steps over dubious records, for instance, penalizing even more the overall performance of the application.

AtyImo is originally built over Spark and runs in a distributed fashion. In this work, we have extended AtyImo to allow for the execution in a hybrid way over heterogeneous platforms composed of accelerators. Thiw new version is called AtyImo-H. If the distributed pipeline is used, AtyImo-H follows the conventional execution flow using the Spark-based solution to transparently distribute parallel tasks across the CPU cores of different computation nodes. If the hybrid pipeline is used and coprocessors are available on that specific compute node where the application runs, it invokes a subroutine that executes in parallel using multicore or manycore parallel API calls.

The proposed solution operates in two distinctive ways: deterministic and probabilistic comparison. In the deterministic comparison, attributes are exactly matched. Information systems that manage benefits often use a unique identifier that can be used as a key for integration between these separate records. However, due to the lack of a common and unique identifier, the integration between administrative and health datasets is

achieved through probabilistic routines using a set of linkage keys, such as name, mother’s name, date of birth, gender etc. The routines implementing the probabilistic linkage were built after an exhaustive statistical study of the characteristics of each dataset involved and the understanding of the relevance of each attribute for the pairwise comparison step. As a result of the linking routine, a new dataset containing an identification key plus complimentary data from the linked records is generated.

AtyImo-H differs from existing tools in terms of i) the heterogeneity of data domains (diverse information systems) and the volume and dimensionality of datasets; ii) demand for highly-accurate results, since there is no gold standards; iii) efficient parallel and distributed implementation in order to solve technological limitations of previous tools; iv) high-performance implementation for the most time demanding tasks within the data linkage pipeline; and v) cross-platform hybrid execution.

5.2 DATA INTEGRATION MODEL

We originally designed and implemented AtyImo-H as a modular pipeline, encapsulating components for data analysis, data pre-processing (cleansing, standardization, blocking, and anonymization), pairwise comparison and matching decision (PITA et al., 2018).

Prior to linkage, all input datasets pass through a data quality analysis stage which performs data integrity checks. This stage is intended to identify attributes suitable for linkage, considering their coexistence in other datasets, the percentage of missing values (especially from linkage attributes), multiple imputation problems, and their ability to uniquely identify individuals.

The next stage is related to data preprocessing (sometimes referred to data standardization) and involves correcting, normalizing, and standardizing field values to improve data quality. In AtyImo-H, this stage aims to circumvent errors, address missing data through imputation, and turn records from different datasets as similar as possible to provide an accurate and fair comparison.

The program consists of a text-mode interface that receives the initial parameters for the two datasets to be linked, identified as larger dataset and smaller dataset. The identification by size (number of rows) is important to define the directionality and cardinality of integration and to decide the best data partitioning strategy. It is also necessary for the user to define which linkage attributes will be used and how standardization should occur: date transformation, gender transformation, municipality (or state) code or name transformation, among other settings. Using date transformation as a brief example, AtyImo-H can manage different patterns: a) *yearmonthday*; b) *daymonthyear*; c) *day-month-year*; d) *day/month/year*; e) *year-month-day*; f) *year/month/day*; g) *month-year-day*; g) *month/year/day*. AtyImo-H documentation is available with the source code with comprehensive directions¹.

In addition, the user also defines which cut-off points will be used for pair classification and whether the blocking/indexing strategy will be used or not. Other system parameters are also configurable if needed. It is possible to determine, for instance: i) maximum disk and memory size (in GB) for intermediate data on different nodes and utilization of CPU

¹Available: <https://github.com/atyimo-lab/atyimo-h>

cores by computation nodes. The intention with this high customization of AtyImo is to allow its use in different search contexts with guaranteed performance and accuracy.

Figure 5.1 shows AtyImo-H’s dataflow and its main inputs and outputs format. Pre-processing is responsible for the harmonization and normalization of records. The procedures implemented in the preprocessing step should return a new view of the dataset to be linked. The transformation stage is responsible for ensuring two important requirements: i) reduction of the search space (use of blocking, for example) and anonymization of identifiable attributes. By the end of this step, all records will consist of a retrieval code and an associated binary vector. Details for Atyimo-H’s pipeline will be presented later.

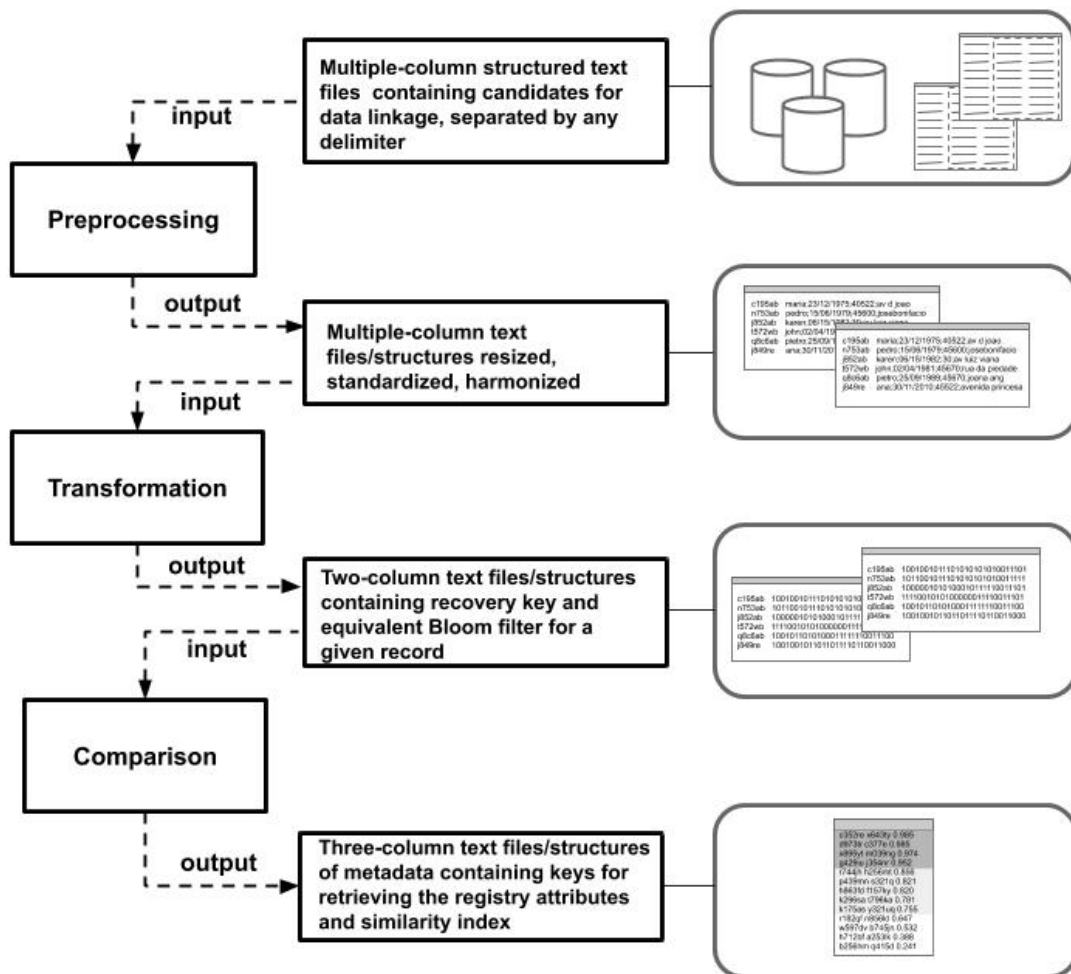


Figure 5.1 Input and output of the three main modules of AtyImo-H’s pipeline.

The data used as input to the application is structured tables in text format, using any delimiter as a field separator. As an output, the tool stores metadata for linked records into three groups:

- positively linked pairs, which are those records classified as “true matches” as they

| | Febri | FRIL | RECLink | AtyImo |
|------------------------------|--------------|-------------|----------------|---------------|
| Reformat values | Yes | Yes | Yes | Yes |
| Remove punctuation | Yes | Yes | Yes | Yes |
| Remove missing values | Yes | Yes | No | Yes |
| Phonetic encoding | Yes | Yes | Yes | Yes |
| Name/Address Standardisation | Yes | No | Yes | Yes |
| Nickname lookup | Yes | No | No | Yes |
| Gender imputation | Yes | No | Yes | Yes |

Table 5.1 Availability of data cleansing functionality across different linkage tools.

present a similarity value above the upper cut-off point;

- non-linked records, which are those records presenting a similarity value below the lower cut-off point;
- “dubious” records, which correspond to records classified as false positive or false negative as their similarity values fall between the upper and lower cut-off points. Usually, this group of records is subject to manual (clerical) review to certify whether they can be considered a true positive or a true negative case. In our case, this manual review is prohibitive (given the size of our datasets), so Aty-Imo can use a machine-learning routine to automatically perform such verification (PITA et al., 2017).

Table 5.2 demonstrates that compared to the three most widely used probabilistic records linkage tools in clinical studies and epidemiological surveillance, AtyImo-H stands out from the rest on issues related to the quality of data processing.

5.3 ATYIMO-H RECORD LINKAGE PIPELINE

During this research, we have developed a solution that implements routines for probabilistic and deterministic linkage of massive datasets, ensuring maximum accuracy, scalability, and performance. Initially, the solution was developed to support the 100 Million Brazilian Cohort, as the existing tools at that time do not allow for managing the expected volume of data, as well as address other technical issues identified in the Brazilian governmental datasets, such as bespoke data cleansing and standardisation.

The linkage solution should be able to handle large volumes of heterogeneous datasets, circumventing poor quality records filling, missing identifier keys, and so on. To solve this problem and provide a scalable and most accurate integration, a modular pipeline solution was implemented, isolating the main steps that make up the integration into independent blocks performing pre-processing, transformation, pairwise comparison, classification and record retrieval.

5.3.1 Application pipeline details

This section presents the description of the modules implemented in AtyImo-H, with emphasis on the core parts: preprocessing, transformation, comparison, sorting and record retrieval (Figure 5.2). The methods that will be taken into account in subsequent chapters are highlighted in a darker shade of gray. It is important to register that other two steps – data analysis and accuracy assessment – are executed before and after these core modules, respectively.

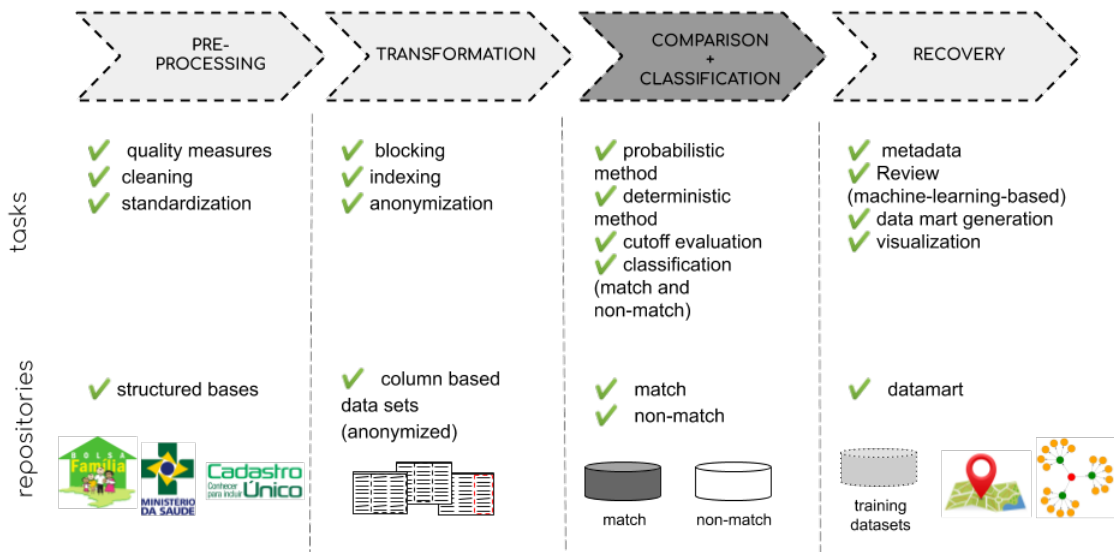


Figure 5.2 Execution pipeline for probabilistic record linkage.

Since it is not possible to establish a gold standard – how many individuals from the social programmes (CadUnico or PBF) will link to any public health care database –, it is necessary to check the completeness of the variables that will be used in the comparison step, as well as to verify the integrity of the data to determine which variables are more appropriate and which could degrade the accuracy of the method. Besides, this analysis is critical to establishing the most appropriate cut-off points (similarity index) for a specific correlation. For these steps, traditional statistical analysis tools such as Stata or R were used.

5.3.1.1 Pre-processing The first step that makes up the linkage pipeline is data pre-processing, responsible for the harmonization and normalization of records. First of all, the datasets pass through cleansing and standardization routines comprising encoding harmonization, removal of special characters, elimination of duplicate records, uniformization of date representations, formatting for expected codes values (such as municipalities), imputation of missing values, etc.

The implementation of blocking is optional and depends on the size of the input file. In practice, if the dataset is small enough to allow for unblocked linkage and if the

hardware infrastructure is suitable, this approach should be used as it ensures better results. However, in our case studies, we have used blocking due to the size of the datasets involved. Among the various blocking approaches we tested, the one with better results was the *predicate blocking*. The procedures implemented in the preprocessing step return a new view of the datasets to be linked, which are represented as files (in case of blocking) or a single, error-free and small file since it contains only the information needed for comparison and information required for retrieving the records in the original files.

5.3.1.2 Transformation The transformation stage is responsible for ensuring two important requirements: i) speed for pairwise comparison, and ii) anonymization of attributes capable of identifying the records. For this, the attributes selected for comparison are mapped into a binary vector using the Bloom filter method. Strings are decomposed into bigrams and each bigram is mapped to a specific vector position through hash functions. At the end of this step, all records will consist only of a retrieval code and an associated binary vector. Thus, this approach favors anonymization and generalizability, as it is possible to perform the comparison step in computing environments with few security resources.

5.3.1.3 Comparison The comparison step can be done deterministically and/or probabilistically, depending on the data being linked and the presence of common key attributes. The user can also set up which method she wants to use. In the probabilistic approach, each pair of records in the search space is submitted to a function that calculates the similarity index between them. The Sørensen-Dice Index was used as an approximation to evaluate how similar two pairs are. If two datasets A and B are compared, all pairs (a, b) obtained from the $A \times B$ product will have an associated similarity value. Therefore, it is necessary to eliminate duplicate records from a , first considering that there are no duplicate records in the original dataset (they were deleted in the preprocessing step) and also considering that the higher similarity index is the most likely candidate to be a true pair. At the end of this step, all pairs and their respective similarity indexes are stored.

5.3.1.4 Sorting While the comparison step receives views of two anonymized files as input and returns the similarity index of each pair, the sorting step seeks to determine whether that pair is a true pair or a false pair. It is also possible to establish a gray area containing “dubious” records that can be recheck if their similarity indexes are not sufficient to determine if they are true or false. At the end of the process, pairs rated as true are stored for later retrieval.

5.3.1.5 Record retrieval Finally, the record retrieval step uses peer codes and the similarity index to retrieve the attributes of interest for the search and ensure an efficient view of linked records that is sufficient to meet business rules.

Figure 5.3 illustrates the execution pipeline of AtyImo-H designed on a heterogeneous

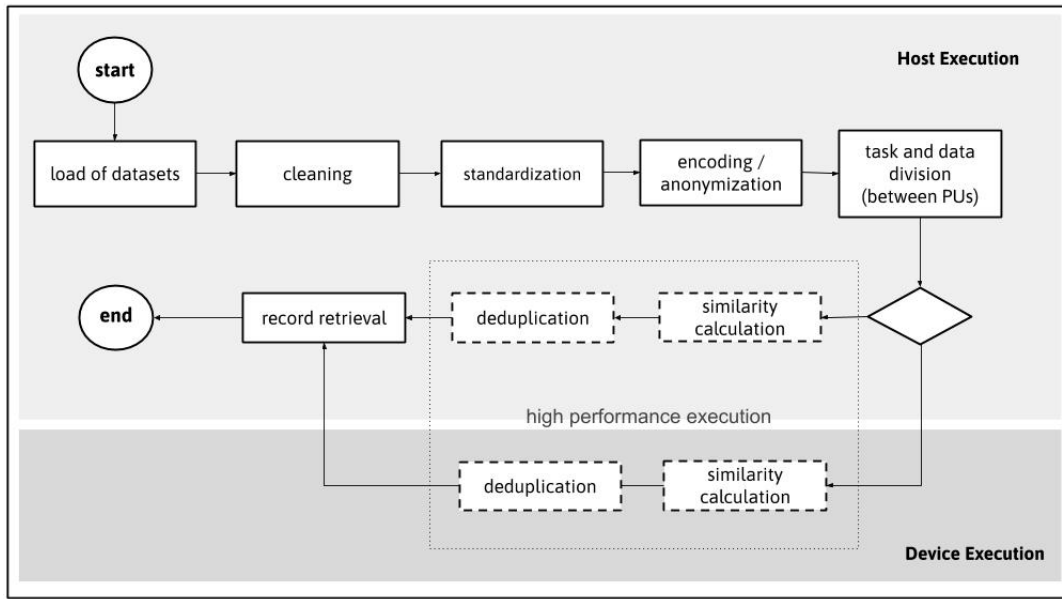


Figure 5.3 Execution pipeline for hybrid model that performs probabilistic linkage on heterogeneous platforms.

computing platform with direct host-device communication. We developed a methodology for splitting the traditional linkage pipeline at the comparison step to provide timely responses for massive, scalable datasets.

This chapter describes in detail the progress of the solution that implements a probabilistic record linkage method, demonstrating through pseudo-codes the development of several approaches associated with their underlying processing units.

PARALLEL MODELING FOR RECORD LINKAGE

This chapter presents considerations for data parallelization and partitioning in heterogeneous computing environments composed of CPUs processing nodes and accelerated by graphic processors GPUs. Initially, we present the strategies to implement multi-core and manycore parallelism to achieve the performance requirements of probabilistic record linkage comparison. We highlight the techniques responsible for minimizing the communication and synchronization overhead between specific PUs. Besides that, we also describe the main algorithms developed in each phase that make up the probabilistic record linkage comparison.

6.1 PROBABILISTIC RECORD LINKAGE

As discussed before, one must carefully evaluate the use of Big Data frameworks when developing solutions with high demand for performance and when response time is crucial for the decision-making stage. If, on the one hand, such frameworks guarantee transparent management of the distributed execution infrastructure, its analytical operations incur in a massive execution time slow down, being an order of magnitude (or more) slower when compared with native parallel implementations (ANDERSON et al., 2017). Because of that, there is a need to circumvent the performance gap for Big Data applications without losing sight of security, fault tolerance, and scalability issues that are also application requirements that must be considered. According to (SINGH; REDDY, 2015) when deciding which platforms to use, one has to investigate, among other things: i) how quickly the application must provide results; ii) how big is the data to be processed; iii) how many iterations will be performed on the same data; iv) what is the scalability limitation for the current solution; v) what are the data transfer pattern/rate and memory footprint expected for this specific workload.

The type of workload considered in this paper can be classified as *in-memory*. In traditional scaling models, it is assumed that there are always two portions of the workload:

a *serial portion* with fixed execution time and constant internal scaling, that is commonly identified as initialization phase; and a parallelizable portion (or external scaling) that grows together with the workload. However, when dealing with data-intensive workloads, both portions (serial and parallelizable) increase at the same proportion, which is known as *in-proportion scaling* (LI et al., 2019). This investigation focuses on in-memory data-parallel scientific application, characterized by a divisible computational workload. It is also expected that this computational workload is proportional to the size of the input data. The partitioning approach we have applied was a static load-balancing method based on functional performance and historical data from previous calibration runs.

Data-intensive tasks built with native parallel implementations, usually demand accelerated responses (to support decision-making process) and require the adaptation of solutions considering this type of workload to handle data while in the main memory without incurring in overheads. Writing to and reading from disk, in this case, degrades the performance of such applications and should be avoided. Parallel optimization for data-intensive problems is limited by several aspects, including: i) the serialization of fine-grained parallel tasks when executing in multicore PU; ii) the amount of local memory available in each PU considered for performing parallel tasks; iii) the data locality and inefficiency of CPU cache hierarchy; and iv) data transfer overhead between host and device.

In a practical context, developing a parallel solution to a data-intensive problem with the premise of generating timely results compatible with decision-making by managers is not a trivial task and involves several obstacles. The deduplication of intermediate results from the different processing units, for example, is another bottleneck since it requires a synchronization of all PUs involved. Therefore, deduplication can only start after the comparison of all pairs has ended. Another problem is that the most robust computing infrastructure for timely responses (complex cluster environment) may not be widely available to all communities. Even the industrial and scientific community has heterogeneous or custom hardware (*commodity*) to perform complex algorithms.

Ideally, applications requiring high-performance solutions would be targeted at the hardware that could scale in parallel in a shared and distributed context as the problem size increase. In real-world, applications could benefit from a multiplatform solution that runs not only on different processing units (taking advantage of the processing power of specialized devices, if they exist, as well as of horizontal and vertical scalability), but also with different programming paradigms, better exploring the hardware characteristics, since offloading computation from a framework like Spark to native C running parallel threads can accelerate user-defined functions as discussed by (ANDERSON et al., 2017). The efficiency of this choice is due to the fact that it is known that the performance of a Big Data application is directly associated with the characteristics of its workload and the nature of the parallel tasks (ALKATHERI; ABBAS; SIDDIQUI, 2019).

A significant advantage of Big Data frameworks is the ability to move computation to the data avoiding data movement costs (such as energy) since data movement corresponds to a significant fraction of 20% of the total workload energy in a memory-bound application, as discussed by (WANG; SCHMIDL; MÜLLER, 2015). However, one can develop solutions with a minimum data transfer, considering intra-node optimizations strategies,

also motivated by the low cost and ever-increasing capacity of DRAM. In practice, solutions able to take advantage of remnant heterogeneous hardware, which would otherwise be idle, and offering adaptable alternatives to accelerate applications high-performance is considered beneficial.

6.1.1 Cross-comparison

Concerning the computational demand, the most critical step in the whole process of probabilistic linkage is pairwise comparison, which requires high availability of main memory and parallel processing resources. Since the complexity of this algorithm is $\mathcal{O}(n \times m)$, the number of iterations and the size of intermediate data that needs to be stored grow quadratically. This characteristic, common to many *Big Data in-memory* applications, is a limiting factor in the use of traditional linkage technologies, leading developers and designers to use massive parallelism architectures and tools, as well as the development of specialized software to ensure scalability and optimized runtime.

The comparison step involves checking all possible pairs to decide whether or not they refer to the same real-world entity. For probabilistic record linkage comparison step an independent task can be defined as the similarity analysis that will be performed on a single candidate pair. Each pair can, therefore, be processed separately and its partial result must be stored for future analysis. If no blocking strategy is used, all records in one datasets will be compared with all records in the other datasets. The total amount of comparisons in this case is $(n - 1) \times (n - 2)/2$. Given that in real datasets this number can be considerably large, it is necessary to build an execution infrastructure capable of distributing tasks across available hardware using the advantages of distributed and parallel programming to generate timely responses.

Algorithm 1 demonstrates a simplified schema of comparing all records in two datasets (A and B) represented by attributes of the same name. The sequential routine inputs are: datasets A and B , $nlines_a$, indicating the number of lines for dataset A , $nlines_b$, indicating the number of lines for dataset B and $size_bloom$, representing the Bloom filter fixed size that comprises a single record. The outermost loop (line 1) iterates over each line of dataset A , and the innermost loop (line 3) iterates over each position of a single record of dataset A records. Since this process occurs after the Bloom filter transformation step, all records have the same size. Each record from dataset A is then stored in a temporary vector $bloomA$ (line 5). While the $bloomA$ vector keeps in main memory the first element of dataset A , the loop described in line 7 iterates over all records from dataset B . The innermost loop (line 9) iterates over each position of a single record of dataset B records, storing its elements in a temporary vector $bloomB$ (line 10) and performing the comparison for the already stored pair of vectors by invoking the procedure *get_dice* (line 11). This procedure returns a similarity index based on the Sorensen Dice coefficient. The procedure described by *deduplicate_dice*(line 12) maintains only the highest Dice index value found among all records compared in the search space of dataset B . Finally, *store_dice* values (line 16) stores the dice value for a given pair.

Algorithm 1 Sequential probabilistic linkage routine.

Input: $*A, *B, nlines_a, nlines_b, size_bloom$ in

Output: $*dice_array$ out

Initialisation : $bloomA \leftarrow bloomB \leftarrow \emptyset$;

- 1: **for** $i = 0$ to $nlines_a$ **do**
- 2: $id_bloomA \leftarrow A[i * (size_bloom + 1)]$;
- 3: **for** $j = 1$ to $size_bloom$ **do**
- 4: $bloomA \leftarrow A[i * (size_bloom + j)]$;
- 5: **end for**
- 6: $dice \leftarrow 0.0$;
- 7: **for** $k = 0$ to $nlines_b$ **do**
- 8: $id_bloomB \leftarrow B[k * (size_bloom + 1)]$;
- 9: **for** $l = 1$ to $size_bloom$ **do**
- 10: $bloomB = B[k * (size_bloom + l)]$;
- 11: $cur_dice = \mathbf{get_dice}(bloomA, bloomB)$;
- 12: $dice_array = \mathbf{deduplicate_dice}(cur_dice, dice,$
- 13: $id_bloomA, id_bloomB)$;
- 14: **end for**
- 15: **end for**
- 16: $\mathbf{store_dice_values}(dice_array, i)$
- 17: **end for**

6.1.2 Approximate similarity measurement

Algorithm 2 demonstrates an implementation that performs the approximate-based similarity calculation, taking as input a pair of records transformed into Bloom filter format and generating the respective similarity index for that specific input pair. We rely on the *Sorensen Dice* index to give an approximation measure between two sets of data. The function takes as input $bloomA$ and $bloomB$ that hold one record from dataset A and B , respectively. Both attributes are already coded into Bloom filters by this moment. $size_bloom$ is also an input parameter for this procedure. Variables a (line 3) and b (line 9) count the amount of 1's contained in $bloomA$ and $bloomB$ vectors, respectively, while h (line 5) counts how many 1's occurred at the same index position for both vectors. The measurement calculated by line 12 is the basis of *Sorensen Dice* equation described by Equation 2.1. Algorithm 3 demonstrates a naïve implementation of the deduplication procedure considering a single pair (cur_dice and $dice$) as input data. This procedure ensures that the Dice attribute always stores only the highest index value for a specific attribute of dataset A in the search space of dataset B. It is important to notice that the algorithm works for single blocks in practice. If the comparison's directionality remains the same $1 \Rightarrow 1$, and dataset A is considered the reference, additional synchronization steps are necessary to eliminate duplicates.

Given the number of iterations associated with the method, it can be seen that the *sequential* routine naturally requires several cycles (depending on the size of the files involved) to finish its execution. However, a reasonable solution is to explore the in-

Algorithm 2 Similarity calculation routine.

Input: **bloomA, *bloomB, size_bloom* in

Output: *cur_dice* out

Initialisation : $a \leftarrow b \leftarrow h \leftarrow dice \leftarrow 0.0$;

```

1: for  $i = 0$  to  $size\_bloom$  do
2:   if ( $bloomA[i] == 1$ ) then
3:      $a ++$ ;
4:     if ( $bloomB[i] == 1$ ) then
5:        $h ++$ ;
6:     end if
7:   end if
8:   if ( $bloomB[i] == 1$ ) then
9:      $b ++$ ;
10:  end if
11: end for
12:  $cur\_dice \leftarrow ((h * 2.0) / (a + b)) * 10000$ ;

```

Algorithm 3 Deduplication routine.

Input: *cur_dice, dice* in

Output: *dice* out

```

1: if ( $cur\_dice > dice$ ) then
2:    $dice = cur\_dice$ 
3: end if

```

intrinsic parallelism of the interactions, distributing each pair of records for independent comparison using the available parallel threads.

6.2 HETEROGENEOUS MODELLING

This section proposes an approach for the coordinated use of multicore and manycore processors considering our application’s specificities. We took as starting point performance limitations of the current Spark-based implementation (AtyImo) and proposed some improvement opportunities by applying explicit parallelism and controlling mechanisms for data movement, communication, and synchronization in a heterogeneous environment.

6.2.1 Linkage pipeline adapted to a hybrid model

The module that performs probabilistic comparison starts on the *host* (CPU), where all input data is loaded. The execution remains on CPU until the data is partitioned and transferred to the device’s global memory space and until an explicit instruction that calls a device execution (GPU kernel) be invoked. All GPU-accelerated functions are also executed in CPU. The workload division is an input parameter for comparison main program. Being customizable, the user can easily specify parameters related to the workload, application, and hardware characteristics. To identify the ideal input

parameters, we consider a calibration step that adjusts the SP and AP values. As System Parameters (SP), we considered i) the number of OpenMP threads, ii) the threads/block ratio, and iii) GPU's global memory. Finally, one, none, or several calls to the parallel kernel are expected to be made. The strategy used to define the tasks' granularity and the data partitioning method will be discussed later.

The GPU accelerated linkage module (that performs cross-comparison, similarity measure and deduplication) requires the transfer of a large amount of data from the main memory (CPU) to the GPU global memory. In the approach presented by this work (and if there is a significant amount of work to justify the time spent on the transfer), the host seeks to maximize the amount of data transferred to the GPU at once. Nevertheless, in real cases, the size of datasets can be considerably large, depending on the number of attributes used to compose the Bloom filter, making the overall memory size of GPUs a limiting factor for application scalability. In these cases, it is necessary to use an adaptive approach that keeps the remaining tasks waiting for an available PU and continue sending them to be processed in the GPU if the performance gain is still guaranteed. The application and study of this type of optimization are outside the scope of this work. Otherwise, the tasks will be executed on the CPU. By the end of GPU-assigned instruction execution, a new communication step is required to transfer the intermediate data back to the host so that the linked pairs can be deduplicated and retrieved. Deduplication between GPUs is also an improvement that enhances overall performance. Since at the end of the execution pairwise comparison each GPU has a copy of the dice vector, instead of GPUs sending their intermediate results to be deduplicated by the host, we use the existing high-speed communication between GPUs to speed up partial deduplication, so that the host is in charge of deduplicating only two sets: its own result and the result of GPUs calculation.

It is important to consider that the GPU will produce intermediate data that must be kept in a shared global memory. These intermediate data in addition with the input and output data sent explicitly by the routines invoking the kernels represent the actual GPU memory consumed by the application and can be defined as *GPU memory footprint*. The fixed size of the data transfer and the expected performance can be previously estimated to establish the best scheduling strategy between processors and coprocessors, considering that the memory demand of a task (*footprint*) or application is known, i.e. the input, output and intermediate data generated by parallel kernels have a predefined size.

6.2.2 Data load and granularity

Parallel *multithread* algorithms have their performance defined by the granularity and distribution of data and the asynchrony of their communication. Fine granularity operates on small portions of a general problem, which facilitates data breakdown between PUs. Ideally, for such a high parallelizable environment, it is convenient to avoid serialized instructions as much as possible and force data parallelism by building applications able to take advantage of a large number of parallel threads.

Therefore, in a finer-grained record linkage implementation, it would be appropriate to represent each record through a set of identifying attributes independently. However,

this strategy incurs two significant deadlocks. The first concerns the vulnerability in the identified information anonymization, as it facilitates *dictionary attacks*, especially for short attributes such as gender or date of birth. The second concerns the storage required to a different Bloom filter for each attribute in a record. In our approach, we merge several attributes in a single Bloom vector to avoid *dictionary attacks* while keeping record linkage optimal accuracy. The Bloom filter efficiency was discussed by ~(PITA et al., 2018) and is out of the scope of this research.

Algorithm 4 Host environment data load.

Input: *percentage_each_gpu, qtd_gpus, nlines_a, nlines_b, size_bloom, size_bloom* in

Output: **A_GPU, *A_CPU, B* out

```

1: pu_edges = get_pu_edges(nlines_a, qtd_gpus, percentage_each_gpu)
2: nlines_a_cpu = pu_edges[(0 * 2) + 1] - pu_edges[0 * 2]
3: for i = 0 to qtd_gpus do
4:   nlines_a_gpu[i] = pu_edges[(i + 1) * 2 + 1] - pu_edges[(i + 1) * 2]
5: end for
6: A_CPU = allocate_host_memory(nlines_a_cpu * size_bloom)
7: for i = 0 to qtd_gpus do
8:   A_GPU[i] = allocate_host_memory(nlines_a_gpu[i] * size_bloom)
9: end for
10: dice_cpu = allocate_host_memory(nlines_a * size_bloom)
11: dice_device = allocate_host_memory(nlines_a * size_bloom)
12: load_file_a(file_dataset_a, A_CPU, pu_edges[0 * 2], nlines_a_cpu)
13: for i = 0 to qtd_gpus do
14:   load_file_a(file_dataset_a, A_GPU[i], pu_edges[(i + 1) * 2], nlines_a_gpu[i])
15: end for

```

We decided to maintain an intermediate granularity considering that all attributes are represented within the same bit vector, on which some sequential instructions will be executed. Data blocks transferred between different devices, in turn, consider a pre-determined grouping of these records. In this sense, it is necessary to carefully consider the transfer time between the host (processing unit where the application is started and the data is loaded) and the other PUs that make up the environment. In principle, it is preferable to store all records in a one-dimensional dataset (*A* and *B*) in contiguous memory. This facilitates data transfers between PUs when a heterogeneous execution approach is required on a heterogeneous platform.

The main program takes as input parameters: i) the GPU grid partitioning information, ii) the relative path for datasets A and B, iii) the definition of CPU threads, iv) the workload percentage assigned to each GPUs, and v) the number of GPUs involved in the computation. Algorithm 4 describes the data load step performed at the beginning of the execution. The *pd_edges* (line 1) vector takes *nlines_a*, *qtd_gpus*, and *percentage_each_gpu* as input and divides the dataset *A* for each UP identifying the upper and lower limits by the line number. The first two elements of *pd_edges* vector (if filled with valid values) identifies the limits of dataset *A* assigned to CPU while the following pairs (if filled with

valid values) identify the limits of A assigned to GPUs. Lines 6 and 8 carry out the matrix's memory allocation that will store subdivided A records in the host memory. Lines 10 and 11 allocate host memory for the Dice vector that will store the result of parallel computation corresponding to each pair's similarity index. Lines 12 and 14, in turn, load the input files into their respective matrix.

6.2.3 Data partitioning

In heterogeneous platforms, data transfer and consequently data locality are the major degradation factors for scalability, performance, and energy efficiency (issues related to energy efficiency are not part of the scope of this research). In this sense, it is extremely important to devise the best strategy for allocating parallel hardware resources when the workload is known, as well as to adapt this workload to the hardware, identifying the best task distribution (based on their characteristics) to the underlying hardware.

Given this, one of the open research challenges in this area is to provide optimal load balancing across all available PUs using data partitioning strategies based on functional performance and historical data. Another related issue is the need to ensure minimal overhead of communication channels and data synchronization by implementing communication strategies suited to the topology over which the application is running.

Since a certain set of different tasks can be performed on both CPU and GPU, the partitioning decision must be made firstly considering which PU provides the best performance and which distribution ensures the best balance. If tasks and instructions diverge, on the other hand, there is probably a more appropriate PU for each. Sequential tasks, for instance, are more appropriate to run on CPUs while highly parallel and fine-grained tasks are more appropriate for GPUs. In some cases, characteristics of the application itself are decisive factors to explain why the allocation of a task to a specific PU is prohibitive, such as when the memory demand of a task (*footprint*) exceeds the PU capacity.

Initially, considering that there is no previous information about the theoretical performance of each processor and considering that all instructions are identical, the tasks that make up the probabilistic comparison step will be performed either on the CPU or GPU without a preferential PU. In this case, what determines the dataset partitioning strategy is the available memory on the GPU and the combined performance of all PUs, obtained through the total execution time of this step.

Figure 6.1 describes how data and tasks are partitioned and distributed to multiple PUs. To maximize the performance of a given application, tasks and data should be evenly shared among processing units from the same type. This is essential to balance the workload between homogeneous PUs. Given two inputs (A and B), each processing unit PU_i has a partition of A , called A_i and a copy (or a vision) of B . Each PU_i performs the tasks related to parallel and sequential steps for cross-comparison, similarity measurement and deduplication needed for record linkage. Figure 6.1 (a) shows a high level model for data and task division considering three processing units using PCI-e to transfer all intermediate blocks of linked pairs and performing deduplication in CPU. In that first example, the deduplication of GPU portion will only start after the CPU finishes its execution. Figure 6.1 (b), on the other hand, shows a synchronization step

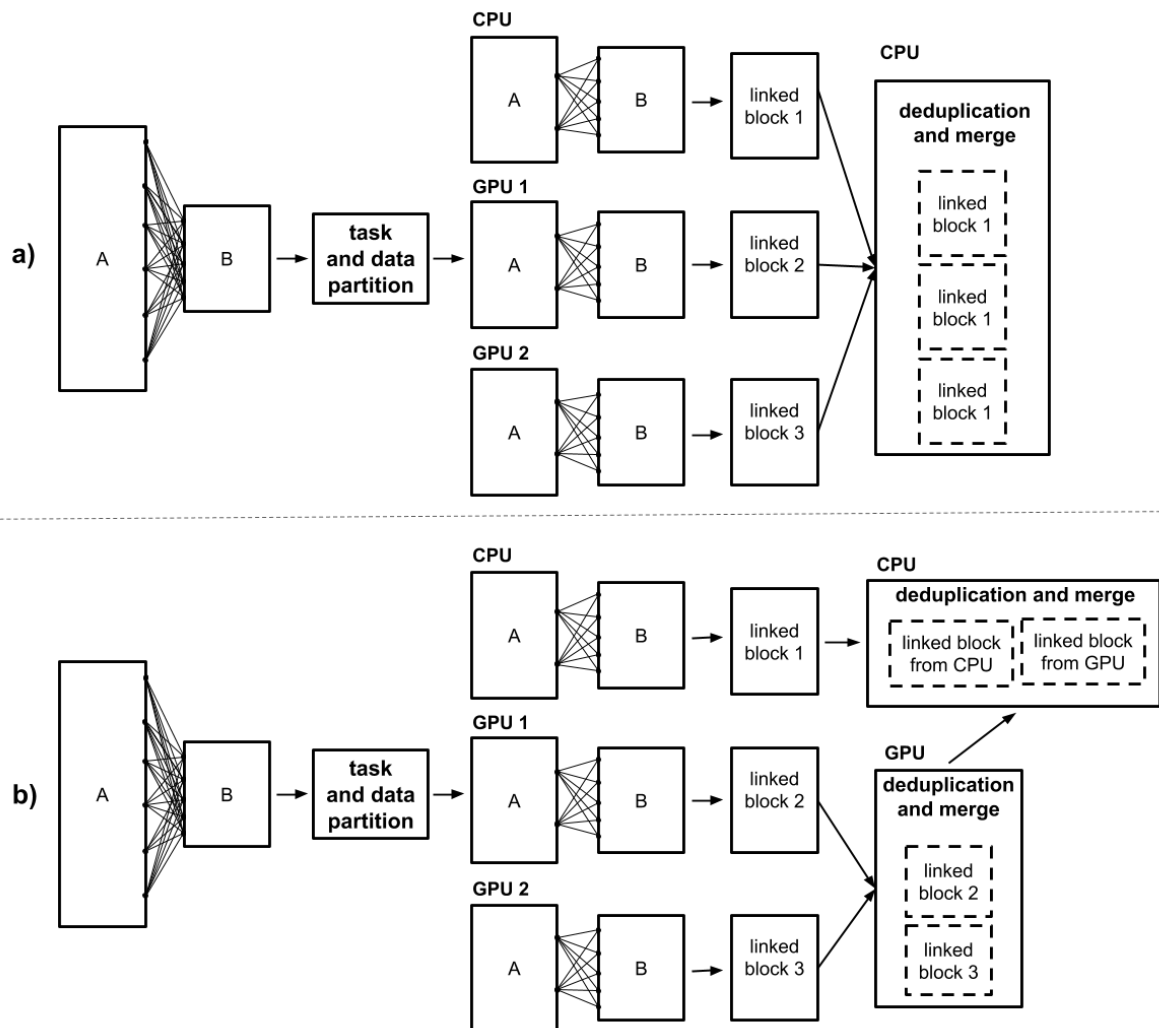


Figure 6.1 High-level overview of parallel linkage and deduplication using 3 Processing Units (one CPU and two GPUs). a) Data partitioning considering deduplication been performed on CPU. b) Data partitioning considering 2/3 of deduplication been performed by GPU.

between GPU *devices* to perform intermediate deduplication of all pairs linked by these processing units. However, this option is only feasible, in terms of performance gain, if there is a NVLINK interconnecting the GPU devices.

Figure 6.2 illustrates a parallel allocation and division scheme between PUs considered on the heterogeneous platform in a scenario where the entire workflow is loaded into the host and special threads are allocated exclusively to coordinate data movement (transfer and synchronization) between *host* and *device*. In the empirical tests presented in this text, we always used an heterogeneous environment composed of one host PU and two devices (two coprocessors and a host). As suggested, it is possible to take advantage of direct communication between two or more devices, considering the existence of a higher speed link instead of a standard PCI-e bus. This communication is useful for sharing

among GPUs the similarity indexes obtained from comparing each pair and eliminating duplicates, keeping only the highest indexing pair (or another strategic decision making).

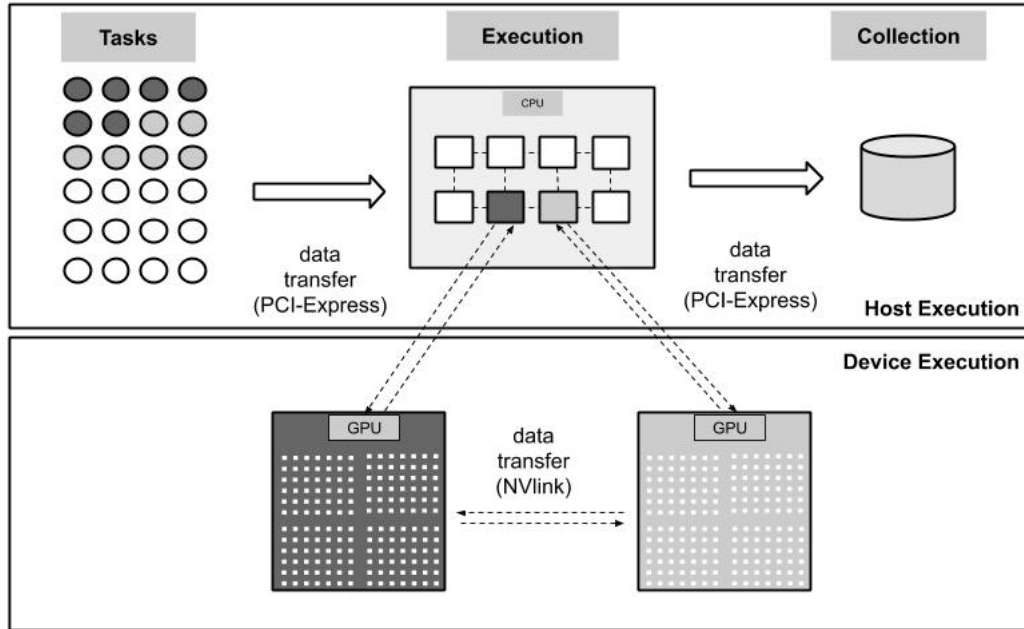


Figure 6.2 Data transfer scheme considering available PUs on the heterogeneous environment.

6.2.4 Multicore parallel probabilistic comparison

To exploit the existing parallelism in multicore processing units, the first parallel approach implemented aimed at to distribute the workload assigned to the GPU between the parallel threads of the physical and logical cores of a CPU processor. Algorithm 5 describes the parallel call of the *multicore_comparison* function (line 6), which is invoked by each parallel thread (created by the constructor of line 3) in an approach that implements coarse granularity over the input data. Considering that multicore processors have fewer physical cores compared to graphics card processors and also considering the large number of records in the datasets used as input, this approach implies higher serialization of the instructions executed for each thread. Line 1 defines in the *quantum* variable the fraction of the largest file assigned to each thread, resulting from the exact division (integer) of the number of records and the number of parallel threads that was defined as system parameters and are available for computation. Line 2, in turn, defines in the *leftover* variable the number of remaining records so that it can be evenly distributed among all threads that will execute with no more than one record in comparison with the others.

Algorithm 6.2.4 describes the division step for the parallel comparison in which *start* and *end* values are calculated. These values define on which fraction indexes of the data block each thread will act. Algorithm 7 receives as input the *start* and *end* indexes, previously calculated, to iterate over the records and performs the pairwise comparison.

Algorithm 5 Multicore parallelization for CPU portion of dataset A.

Input: $A_CPU, B, nlines_b, pu_edges, threads_cpu, id_pu, size_dice$ in

Output: $dice_cpu$ out

```

  Initialisation :  $*dice\_acc \leftarrow \emptyset$ ;
  Initialisation :  $quantum \leftarrow leftover \leftarrow 0.0$ 
1:  $quantum = pu\_edges[(id\_pu * 2) + 1] / threads\_cpu$ 
2:  $leftover = (quantum - (integer)quantum) * threads\_cpu$ 
3: # start parallel region
4: {
5:  $id\_nested = get\_thread\_num()$ 
6: multicore\_comparison( $A\_CPU, B, dice\_cpu, nlines\_b, id\_nested,$ 
7:  $*dice\_acc, size\_dice, (integer)quantum, (integer)leftover$ )
8: store\_dice\_values( $dice\_acc, id\_nested$ )
9: }
10: deduplicate\_dice\_set( $dice\_acc, dice\_cpu$ )

```

Algorithm 6 Definition of start and end positions for each multicore thread.

Input: $id_nested, leftover, quantum$ in

Output: $start, end$ out

```

  Initialisation :  $*dice\_acc \leftarrow \emptyset$ ;
  Initialisation :  $quantum \leftarrow leftover \leftarrow 0.0$ 
1: if ( $id\_nested < leftover$ ) then
2:    $start = id\_nested * (quantum + 1)$ 
3:    $end = quantum + 1 + start$ 
4: else if ( $id\_nested == leftover$ ) then
5:   if ( $leftover = 0$ ) then
6:      $start = id\_nested * quantum$ 
7:      $end = start + quantum$ 
8:   else
9:      $start = id\_nested * (quantum + 1)$ 
10:     $end = start + quantum$ 
11:   end if
12: else
13:   if ( $leftover == 0$ ) then
14:      $start = id\_nested * (quantum)$ 
15:      $end = start + quantum$ 
16:   else
17:      $start = id\_nested * (quantum + 1) - (id\_nested - leftover)$ 
18:      $end = start + quantum$ 
19:   end if
20: end if

```

All threads will access the same shared copy of A_CPU fraction and dataset B . The $dice_cpu$ vector will be returned to the master thread that will perform the deduplication.

Algorithm 7 Parallel comparison for each multicore thread.

Input: $start, end, A_CPU, B, size_bloom, nlines_b$ in

Output: $*dice_cpu$ out

```

1: while  $start < end$  do
2:    $id\_bloomA = A\_CPU[start * size\_bloom]$ 
3:   for  $j = 1$  to  $size\_bloom$  do
4:      $bloomA[j - 1] = A[start * size\_bloom + j];$ 
5:   end for
6:    $dice \leftarrow 0.0$ 
7:   for  $k = 0$  to  $nlines\_b$  do
8:      $id\_bloomB = B[k * size\_bloom];$ 
9:     for  $l = 1$  to  $size\_bloom$  do
10:       $bloomB[l - 1] = B[k * size\_bloom + l];$ 
11:    end for
12:     $cur\_dice = dice\_multicore(bloomA, bloomB)$ 
13:     $dice = deduplicate\_dice(cur\_dice, dice,$ 
14:       $id\_bloomA, id\_bloomB)$ 
15:  end for
16:  store\_values\_dice( $dice\_cpu, start$ )
17:   $start++;$ 
18: end while

```

6.2.5 Hybrid-parallel probabilistic comparison over heterogeneous environment

Algorithm 8 demonstrates the scheme used to partition the workload between two different PUs. In this case, matrix A will be divided (CPU will take A_CPU and GPUs will take $A_GPU < id_pu >$) and its computation will be in charge of each PU, while the matrix B will be sent entirely to all devices avoiding inconsistencies and duplicated results. The pu_edges vector stores the fraction (initial and final indexes) of the dataset A_CPU and $A_GPU < id_pu >$ associated with each available PU. In this approach a static distribution strategy has been implemented, considering that the performance of each PU can be known *a priori*. One CPU thread is needed to manage each PU involved in the computation; so the execution starts with a number of threads equal to the number of available devices (GPUs) plus one. The threads controlling the GPUs will execute the block code from line 8 to 10 and the remaining threads will coordinate the openMP parallelism, executing the block code from line 4 to 7.

Still considering the workload distribution among the different PUs described by Algorithm 8, $qtd_gpu + 1$ threads are initially created as indicated by line 2. The first thread (thread 0) is tied to multicore execution and is responsible for creating a nested CPU thread set. Thread 0 calls `cpu_execution` procedure and generate a `dice_cpu` vector (line

Algorithm 8 Definition of threads controlling the workload division for each PU.

Input: *qtd_gpu, *pu_edges* in

Output: **dice_array* out

```

1: omp_set_nested(1)
2: #pragma_omp_parallel num_threads(qtd_gpu+1)
3: id_pu = omp_get_thread_num()
4: if ((id_pu == 0)  $\wedge$  (pu_edges[id_pu * 2]  $\neq$  -1)) then
5:   dice_cpu = cpu_execution(A_CPU, B, nlines_b,
6:   pu_edges, threads_cpu, id_pu)
7: end if
8: if (id_pu  $\neq$  0)  $\wedge$  (pu_edges[id_pu]  $\neq$  -1) then
9:   dice_gpu = gpu_execution(id_pu, A_GPU(id_pu),
10:  pu_edges);
11: end if
12: deduplicate_dice_array(dice_cpu, dice_gpu)

```

Algorithm 9 Definition of the device attached to each GPU thread, device allocation, copy and kernel invocation.

Input: *size_dice, nlines_b, bloom_size, id_pu, B, A_GPU, pu_edges, threads_per_block*
in

Output: **diceGPU* out

```

1: setDevice(id_pu - 1)
2: B_d = allocate_device_memory(nlines_b, bloom_size)
3: A_d = allocate_device_memory(pu_edges, bloom_size, id_pu)
4: dice_GPU_d = allocate_device_memory(nlines_b, size_dice)
5: copy_host_to_device(B_d, B)
6: copy_host_to_device(A_d, A_GPU(id_pu))
7: define_kernel_dimension(pu_edges, id_pu, threads_per_block)
8: call_kernell(dice_GPU_d, A_d, B_d, (pu_edges[(id_pu * 2) + 1] -
9: pu_edges[id_pu * 2]), nlines_b)
10: copy_device_to_host(diceGPU, diceGPU_d2)
11: free_device_memory(B_d, A_d, dice_GPU_d)

```

6) which has the same size as the number of lines of dataset B (*nlines_b*). The other threads are tied to GPU execution and are responsible for invoking GPU kernels. GPU threads call `gpu_execution` procedure and receive `dice_gpu` vector which has the same size as the number of lines of dataset B (*nlines_b*). If there is NVLINK interconnection between the GPU devices, GPU threads will generate only one partially deduplicated `dice_gpu` vector (line 9); otherwise, each GPU thread will generate a different `dice_gpu` vector. In a multi-GPU environment composed of two GPUs, two threads manage the GPUs while the others perform CPU computation. Each processing unit will perform its computation if `pu_edges[id_pu * 2] != -1`, that is, if there is data to be processed for that specific PU.

The computation performed by the GPU are invoked by Algorithm 9. It takes as input the size of one record of the Dice array, the number of records of dataset B, the size of the Bloom filter, the *thread_id* of the specific device, the entire dataset B, a portion of dataset A (assigned for that device), *pd_edges* and the number of threads per block to be launched. For the implementation developed in this work, we use a *size_dice* = 3 to store the indexes of each record of a pair and its respective similarity index. As output, the procedure will generate *diceGPU* array which has the same size as the number of lines of dataset B. Each multicore thread *id_pu* sets its respective GPU device. It is important to notice that, in different environments, GPU devices can be set in a different order than was described by Algorithm ???. We considered that cards were set in an orderly manner starting at 0 (line 1). For each structure that will be copied to the GPU global memory (line 5 and 6), we allocate respective GPU memory (line 2 to 4). After that, we define the grid and blocks dimension (line 7) and finally call the GPU kernel to operate on each dataset portion. A copy back to the host main memory is needed to bring the result (*dice_GPU*).

6.3 DISCUSSION CONCERNING TECHNOLOGIES

The algorithms described in this section refer to source codes implemented using the C programming language chosen due to its high portability in multiple environments and comprehensive support offered to APIs and parallel programming paradigms, facilitating the application's maintenance in future steps and promoting code reuse for specific modules. We used the well-known OpenMP API to implement the parallelism, synchronization, and communication of tasks in the multicore execution environment. The codes running in GPU, in turn, were implemented using the CUDA API, and the most recent tests used the most updated versions of all mentioned APIs.

Despite this, throughout this chapter, we attempt to describe the algorithms in a generic way aiming to characterize the solution as technology-independent, as assumed in the research proposal. In future works, one can apply tests with different API.

In this section, we evaluate the Hybrid Record Linkage module considering performance, accuracy and hardware usage. We present the progress of our solution in a chronological arrangement, showing the limitations and improvements for each approach.

HYBRID RECORD LINKAGE EXPERIMENTS

In order to analyze the results achieved from the developed approaches, two important and distinct metrics are taken into consideration. The first concerns accuracy, which is directly influenced by the number of different values that will be used to identify a given record, by the quality of attributes completion, which must be analyzed and improved by the pipeline actions that are performed before the comparison step and finally by the quality of transformation and classification algorithms. The second evaluation concerns the performance delivered by the execution time, scalability and hardware usage. We show in this chapter the relationship between parallel time and sequential time, given by the *speedup* factor (when comparing different devices, *speedup* factor is given considering the ratio between parallel time and the execution time for the other processing unit) and we also present a comparison with an alternative Spark-based execution. Parallel executions, even when using Spark as an execution framework, use a single computational node. This guarantee a fair comparison between native code solution and Spark framework since we deal only with shared-memory parallelism. Issues related to distributed communication are out of the scope of this work.

7.1 ACCURACY ANALYSIS

As a proof of concept, this approach was first applied to a real scenario consisting of a 5-year extraction of the **CadastroUnico (CADU)** socioeconomic dataset¹. These data were linked to mortality data from the **Mortality Information System (SIM)** dataset provided by the Ministry of Health². Whereas these two systems were not previously designed for integration, there is not a unique key capable of promoting reliable deterministic correlation. In this way, the probabilistic linkage becomes a requirement.

¹<http://www.caixa.gov.br/cadastros/cadastro-unico/Paginas/default.aspx>

²<https://www.saude.gov.br/saude-de-a-z/mortalidade>

We have been linking Brazilian databases for seven years so far, and have relied on past works to validate our results (PITA et al., 2018), (BORATTO et al., 2018), (PINTO et al., 2017c), (PINTO et al., 2017a), (PINTO et al., 2017b), (PINTO; BARRETO; BORATTO, 2016), (PITA et al., 2015).

In this research, the accuracy evaluation is influenced by four main factors: i) quantity and quality of the attributes selected for comparison; ii) predetermined size of Bloom filter and the respective number of hash functions chosen for mapping bigrams; iii) reliability of the similarity index; and iv) decision to use or not blocking methods. Accuracy is assessed by *recall*, *precision* and *positive predictive value* (PPV), which refers to the amount of true positives observed in the *matches* set. For this integration, data from the Brazilian states of Sergipe (SE), Santa Catarina (SC) and Rondônia (RO) were used. These states were chosen for having high representativeness of the Brazilian population, for the great divergence regarding the quality of the stored information and the viability for manual verification.

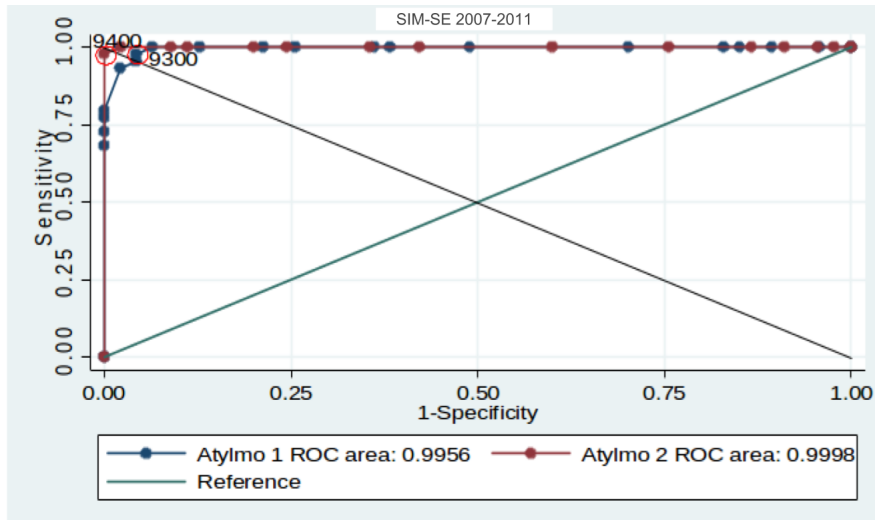


Figure 7.1 Accuracy assessment in the integration of SIM-SE and CADU (cohort) data.

Figures 7.1, 7.2 and 7.1 describe the accuracy (hit rate) given by ROC curves for the probabilistic linkage solution. The maximum value below the curve (Figure 7.3) reached 1.00 with an accuracy of up to 100%, which means that no pair has been classified as false negative or false positive. The minimum value below the curve was 9.9 (Figure 7.2) with an accuracy of 97% with similar recall and precision values.

7.2 PERFORMANCE ANALYSIS

Since the main purpose of this research is the provision of scalable and efficient pairwise comparison routines to be included in the Atylmo-H tool, accelerating the probabilistic comparison stage, this section will be dedicated to the performance analyzes, tests and considerations on specificities and limitations of the approaches developed for the specific domain. Considering the *weak scalability*, the ability of the application to maintain a

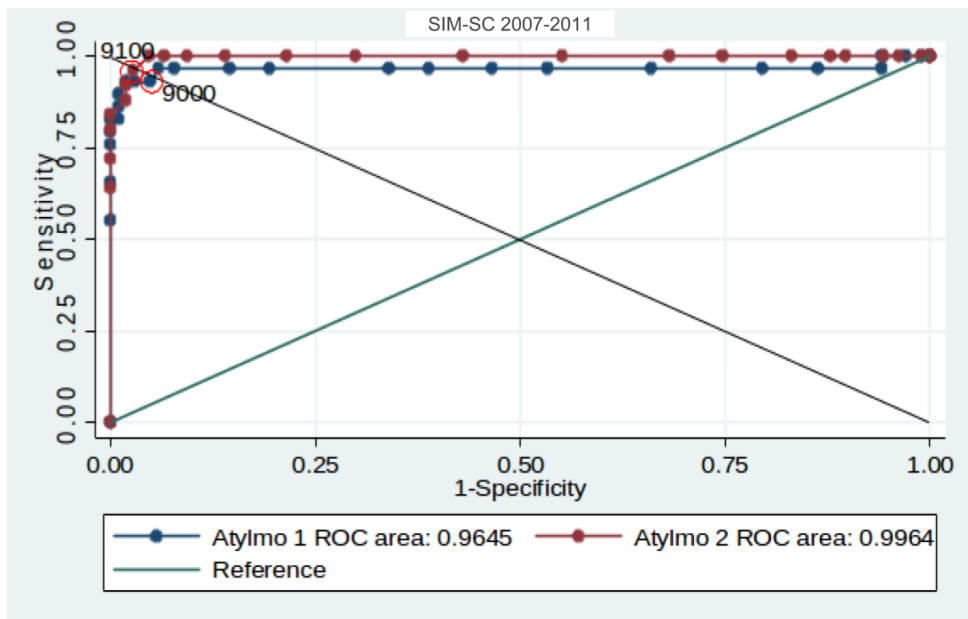


Figure 7.2 Accuracy assessment in the integration of SIM-SC and CADU (cohort) data.

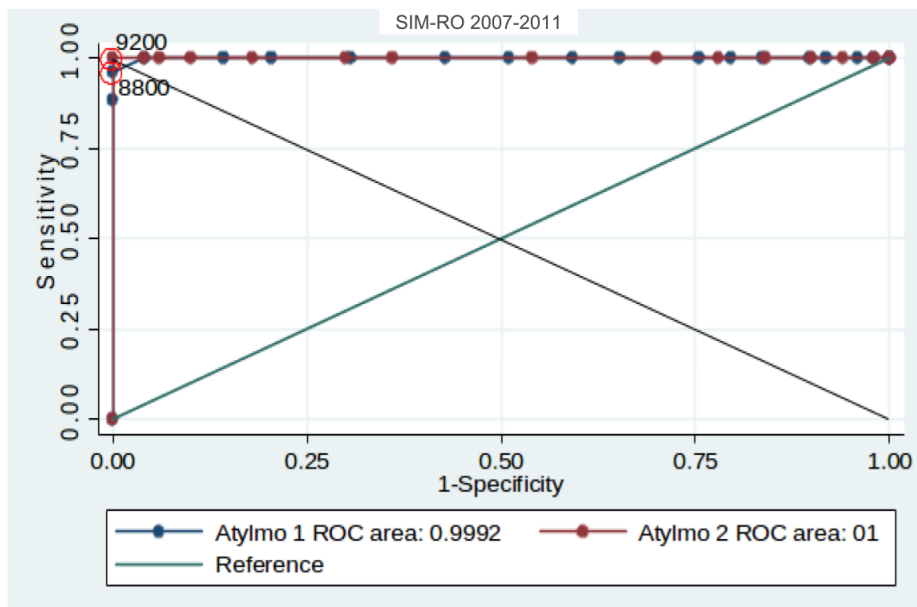


Figure 7.3 Accuracy assessment in the integration of SIM-RO and CADU (cohort) data.

stable performance was analyzed as it increases two parameters in the same proportion: the size of the problem and the level of parallelism/number of processors. In tests of *strong scalability*, on the other hand, we analyzed the ability of the application to achieve better performance as more processing units are added. In general, we considered as scalable an algorithm that is able to adjust the parameters of the system to maintain a satisfactory efficiency and performance as the problem size grows.

7.2.1 Hybrid record linkage up to 20 million records

7.2.1.1 Experimental setup (test purpose) The tests presented in this section were performed on a system comprised of 4 Intel Xeon CPU, 2.93 GHz. Each processor is a quadcore with 24 MB of cache. The DDR3 main memory has a total of 130 GB. The system is also comprised of Two Tesla k40 GPU graphic cards with 28 *Stream Multiprocessors (SM)* and 64 *Stream Processors (SP)*. The CUDA version 4.0 was used. Updated versions were applied in subsequent tests.

Table 7.1 Execution time for different performance parameters (best values in bold).

| s | $w = 45, 45, 10$ | | $w = 40, 40, 20$ | | $w = 35, 35, 30$ | |
|------------|------------------|--------------|------------------|--------------|------------------|--------------|
| | c | $t(s, c, w)$ | c | $t(s, c, w)$ | c | $t(s, c, w)$ |
| 1,000.000 | 30 | 7,90 | 30 | 5,48 | 30 | 5,39 |
| 2,000.000 | 30 | 11,38 | 30 | 7,89 | 30 | 7,36 |
| 4,000.000 | 30 | 17,95 | 30 | 11,69 | 30 | 10,43 |
| 6,000.000 | 30 | 17,95 | 30 | 11,69 | 30 | 10,43 |
| 8,000.000 | 30 | 25,62 | 30 | 22,06 | 30 | 15,68 |
| 10,000.000 | 30 | 26,59 | 30 | 28,95 | 30 | 20,90 |
| 12,000.000 | 30 | 26,87 | 30 | 20,02 | 30 | 19,87 |
| 14,000.000 | 30 | 30,95 | 30 | 29,10 | 30 | 21,89 |
| 16,000.000 | 30 | 40,36 | 30 | 30,30 | 30 | 27,25 |
| 18,000.000 | 30 | 51,83 | 30 | 28,01 | 30 | 26,19 |
| 20,000.000 | 30 | 59,42 | 30 | 37,49 | 30 | 25,12 |

7.2.1.2 Performance analysis In the initial scalability tests we used the parallel version that considers and compares: i) an isolated execution of CPU using OpenMP; ii) one GPU using CUDA; iii) two GPUs and iv) an hybrid approach, using the CPU and two GPUs collaboratively on the heterogeneous platform presented in the subsection 7.2.1.1. The first runs were dedicated to the calibration step. For this, different configurations were used for Algorithm Parameters (AP) and System Parameters (SP). Different thread allocations (1,2,...,30) were tested using the enabled mode for *Intel Hyper-Threading* (ÉTIENNE, 2012). The workload (w) was split into different percentages ranging from 10% to 45% for each GPU. In all experiments, the remaining percentage of the workload was directed to multicore (CPU) execution. The number of linked records

Table 7.2 Comparison of execution times (in seconds) (best values in bold).

| s | CPU cores | 1GPU | 2GPUs | Hybrid |
|------------|-----------|-------|-------|--------------|
| 1,000.000 | 43,94 | 9,15 | 6,71 | 5,49 |
| 2,000.000 | 85,99 | 10,03 | 10,63 | 7,37 |
| 4,000.000 | 162,37 | 15,47 | 17,37 | 10,44 |
| 6,000.000 | 245,09 | 17,79 | 24,54 | 13,18 |
| 8,000.000 | 318,04 | 20,76 | 30,66 | 15,69 |
| 10,000.000 | 402,60 | 25,69 | 32,64 | 17,90 |
| 12,000.000 | 510,74 | 29,44 | 37,94 | 19,87 |
| 14,000.000 | 620,78 | 31,40 | 38,71 | 21,89 |
| 16,000.000 | 682,10 | 35,34 | 40,10 | 25,25 |
| 18,000.000 | 766,58 | 34,65 | 52,81 | 26,19 |
| 20,000.000 | 850,94 | 38,58 | 68,07 | 27,12 |

(comparisons and classifications) in these experiments ranged from 1 to 20 millions as shown by column s. Table 7.1 shows the values used as parameters and the execution times associated with each execution.

Table 7.3 Comparative performance, given by the gain with respect to time of multicore execution (best values in bold).

| s | 1GPU | 2GPUs | Hybrid |
|------------|-------|-------|--------------|
| 1,000.000 | 4.33 | 6.55 | 8.01 |
| 2,000.000 | 8.57 | 8.09 | 11.68 |
| 3,000.000 | 10.89 | 9.44 | 14.62 |
| 4,000.000 | 10.50 | 9.34 | 15.56 |
| 5,000.000 | 14.44 | 10.23 | 17.65 |
| 6,000.000 | 13.78 | 9.99 | 18.59 |
| 7,000.000 | 14.84 | 9.84 | 19.79 |
| 8,000.000 | 15.32 | 10.37 | 20.27 |
| 9,000.000 | 18.53 | 11.53 | 25.03 |
| 10,000.000 | 18.01 | 11.38 | 22.13 |
| 12,000.000 | 17.35 | 13.46 | 25.70 |
| 14,000.000 | 19.77 | 16.04 | 28.35 |
| 16,000.000 | 19.68 | 17.01 | 27.15 |
| 18,000.000 | 21.69 | 14.51 | 29.26 |
| 20,000.000 | 22.05 | 12.50 | 31.22 |

In general, the value of c depends on the size of the problem. However, as in this comparison step, all the instructions used are identical and follow the *SIMD* paradigm, and also considering that there is always a workload big enough to be shared between the parallel threads of a processor. The value of 30 threads was set specifically for this

execution environment. On the other hand, for different s values (and on different platforms), there may be appropriate w settings, given that a different load will be launched to each PU.

Table 7.2 and Table 7.3 show the execution time and the gain with respect to time of multicore execution, respectively, considering each PU presented. The time regarding using multicore threads isolated is given by *CPU colors*. The time for execution in only one GPU is given by the *1GPU* column, while the simultaneous execution time using two GPUs is given by *2GPUs* column. The last column, *Hybrid*, demonstrates the runtime considering the three PUs defined above. The results demonstrate that the hybrid approach achieves superior results compared to running exclusively on CPU or GPUs, especially for large data volumes, reaching a maximum gain of 31.22 times.

For comparison purposes, each execution was performed on isolated PUs (CPU, one GPU, and two GPUs) and then in an hybrid way. Figures 7.4 and 7.5 show the execution times respectively considering all PUs except the CPU (for better viewing purposes). In the experiments presented in this section, the performance gain is obtained considering only the CPU execution time (multicore). Figure 7.6 presents the parallel gain for linkage considering all parallel approaches implemented, including the hybrid one. Figure 7.7, in turn, illustrates the runtime disparity for different workload distributions.

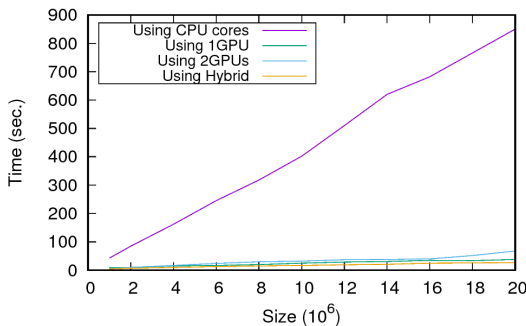


Figure 7.4 Execution time for linkage step considering CPU, one GPU and two GPUs executed in isolation and its comparison with the hybrid approach.

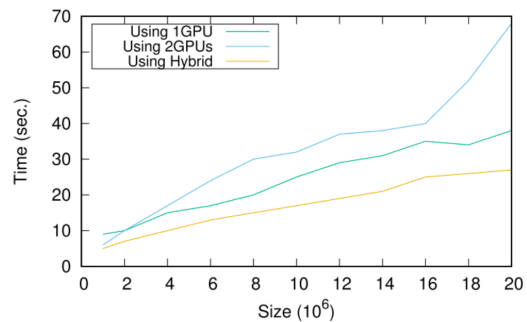


Figure 7.5 Execution time for linkage step considering one GPU and two GPUs executed in isolation and its comparison with the hybrid approach.

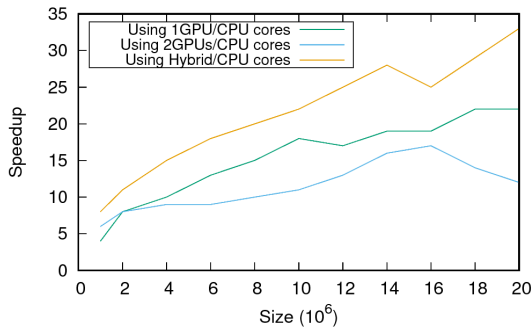


Figure 7.6 Speedup for the linkage step considering each approach implemented.

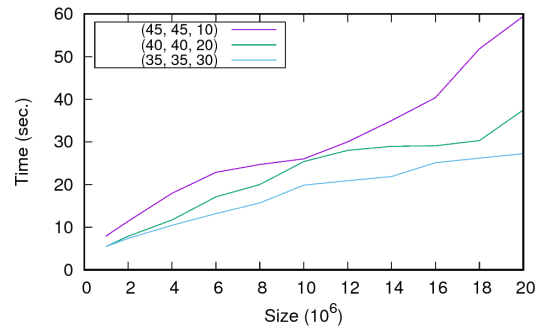


Figure 7.7 Execution time for each workload division between PUs in the sequence: GPU, GPU, and CPU.

The results presented by these experiments demonstrate the superiority of the hybrid approach given by the inclusion of GPU accelerators. It is important to notice when considering the execution time of isolated PUs that the execution using only one GPU has better performance when compared to the use of two GPUs collectively. This fact can be explained considering that, for this size of problem, the cost of data transfer and device management by the CPU (and consequent synchronization required for parallel tasks) can harm overall performance. Subsequent sections will present alternatives to mitigate this effect.

7.2.2 Hybrid record linkage up to 70 million records

Seeking the purpose of comparing a larger number of pairs to enable the probabilistic link of 100 million records (quantity compatible with the size of the 100 Million Brazilian Cohort) without using blocking methods, the work was focused on improving the algorithm concerning dataset load, partitioning, transfer between the available PUs and updating the hardware and software. In this subsection, the evolution of the development and tests related to the hybrid comparison of the probabilistic linkage is presented. To achieve these results, we updated the code to be compatible with CUDA 10 which adds more features, performance improvements, bug fixes, in addition to improvements to the standalone tools for debugging and profiling. Another important improvement concerns the support offered by CUDA 10 for peer-to-peer communication between GPUs using NVLINK: an interconnection with much higher bandwidth. This is a powerful feature, especially for large memory workloads, that allow data to be split across the frame buffer of both GPUs.

7.2.2.1 Experimental setup The tests presented in this section were performed on a GPU node consisting of two sockets Intel Xeon Gold 6148, with a frequency of 2.40GHz; 20 cores in each socket (representing a total of 40 cores in a single node); cache with 28MB; main memory equal to 192GB. The GPU node is composed of two Tesla P100-SXM2 graphic cards with global memory of 16GB, 732GB/s of memory bandwidth, 3584

CUDA Cores and *NVLINK* bus interconnectivity.

7.2.2.2 Performance analysis In this subsection, we describe the performance evaluation results associated with the computing time of the comparison step from *AtyImo-H* pipeline when executed exclusively using the hybrid option. As system parameters (SP), the property that varies is related with the multicore and manycore parallelism. In this context, we have considered: number of OpenMP threads and the ratio of threads/block. As application parameters (AP), the variable properties of the input problem were considered: number of records in the largest dataset and the proportion of the load to be assigned to each PU involved in the computation given by the size of the data stored in the memory of the devices for each problem. input. For the scalability tests, the number of OpenMP threads varied from 1 (serial execution) to the logical thread limit of the processor. Considering the use of hyperthreading, this maximum amount reached 80 threads, while the maximum number of physical threads was 40 (double).

The input problem ranged from 1 million records to 70 million records. The smaller base was fixed at 10,000 records in all experiments. From 1 million to 20 million records, the sizes of the input problem were divided into regular intervals of 2 million records. From 20 million to 70 million the problem sizes were divided into regular intervals of 10 million records in order to allow a better visual and numerical analysis of the results obtained.

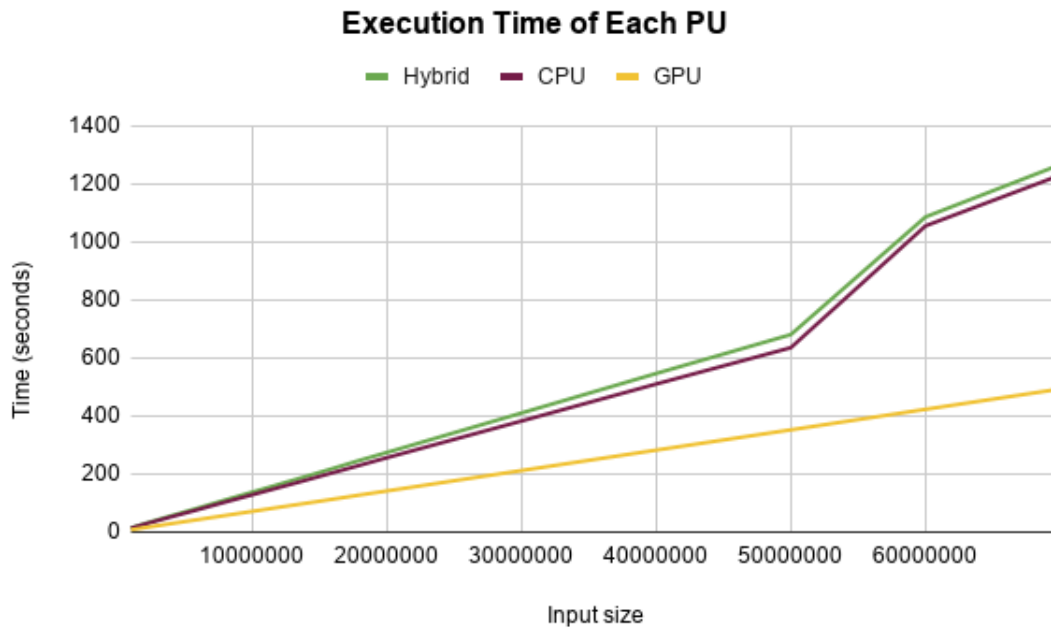


Figure 7.8 Execution time of the hybrid approach and its comparison with CPU and GPU runtime in an environment composed of two P100 GPUs.

In the first step of this study, the execution times of the hybrid approach are demonstrated when the size of the input problem varies. The time of the hybrid approach

| Input Size | Load time | CPU | GPU | Hybrid |
|------------|-----------|-------------|------------|-------------|
| 1000000 | 0.904837 | 13.477709 | 8.338676 | 14.390918 |
| 2000000 | 1.801966 | 27.174393 | 15.50223 | 28.986045 |
| 4000000 | 3.671854 | 52.83951 | 29.365366 | 56.522866 |
| 6000000 | 5.350599 | 77.816935 | 43.368177 | 83.181235 |
| 8000000 | 7.214579 | 103.603251 | 57.443239 | 110.833397 |
| 10000000 | 9.117034 | 128.562706 | 71.442332 | 137.697658 |
| 12000000 | 10.830713 | 153.836007 | 85.693925 | 164.687291 |
| 14000000 | 12.798385 | 179.161539 | 99.532884 | 191.981039 |
| 16000000 | 14.541385 | 204.856424 | 113.555748 | 219.425947 |
| 18000000 | 17.299208 | 230.360265 | 128.040982 | 247.686059 |
| 20000000 | 18.057199 | 256.373002 | 141.854134 | 274.462408 |
| 30000000 | 27.565059 | 382.892001 | 212.334932 | 410.501801 |
| 40000000 | 36.214255 | 510.442144 | 282.638174 | 546.708405 |
| 50000000 | 45.113597 | 635.613141 | 352.563669 | 680.787612 |
| 60000000 | 53.76732 | 1055.390382 | 423.232657 | 1086.24251 |
| 70000000 | 61.164862 | 1230.217209 | 493.336969 | 1266.463188 |

Table 7.4 Execution time of each PU and time spent loading datasets into devices' memory up to 70 million records (time in seconds).

considers the complete execution time of the comparison stage and comprises the execution time of the block that performs loading (reading the data and partitioning), the execution time of the multi-GPU block (for this experiment only two GPUs were considered), and the execution time of the multicore block. The execution time of each of these blocks is highlighted in the graphs and tables so that one can achieve the most realistic understanding of the high-performance hybrid solution and the impact of each block in the total performance.

Figure 7.8 establishes a comparison between the total execution time of the hybrid approach concerning the multicore block that runs on the CPU and the manycore multi-GPU block that runs on two GPUs. We can note that the multicore execution largely limits the execution time of the hybrid approach. Figure 7.9 shows more clearly the performance gain for the multi-GPU block when compared to the block that runs on the CPU. In the smallest size of the problem, the gain fluctuates around 1.6x while in the largest size of the input problem the gain is around 2.49x. An important consideration point is that even with 2.49x of gain, it is not correct to say that transferring the load from the CPU to the GPU would compensate the high CPU execution time since this would generate a greater overhead and a significant increase in execution time. The use of streams also does not benefit applications of this nature, since the instructions executed on the GPU are simple and are completed in a few cycles.

Regarding multi-GPU execution, it is necessary to understand the impact that each step has on the execution time. For this purpose, Figure 7.10 differentiates for each input problem: a) the time spent by the GPU; b) the time spent with transfers between host

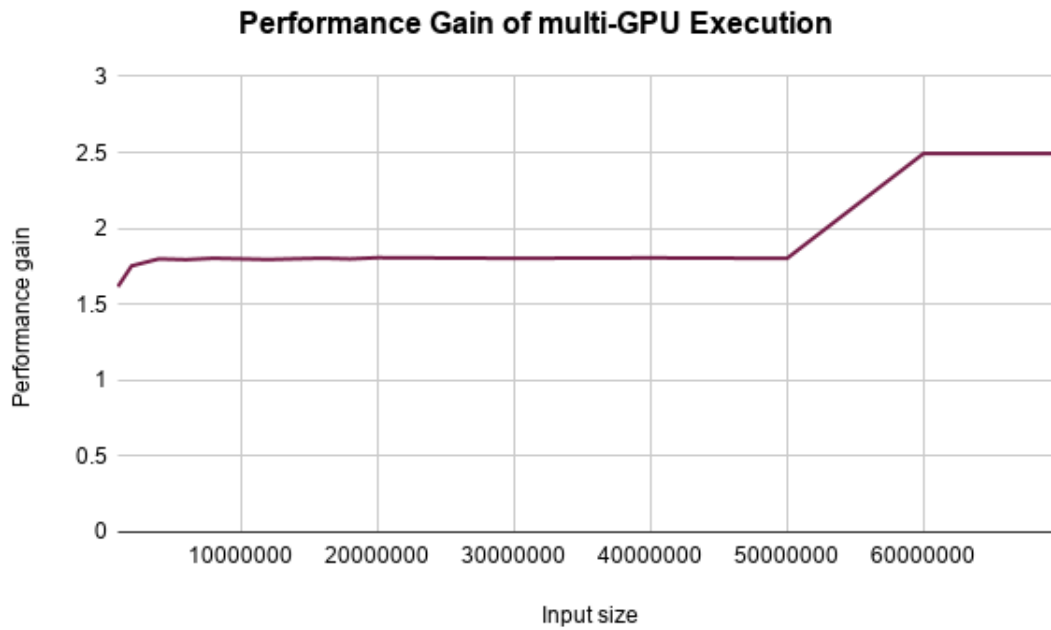


Figure 7.9 Performance gain of multi-GPU execution in comparison with CPU execution time.

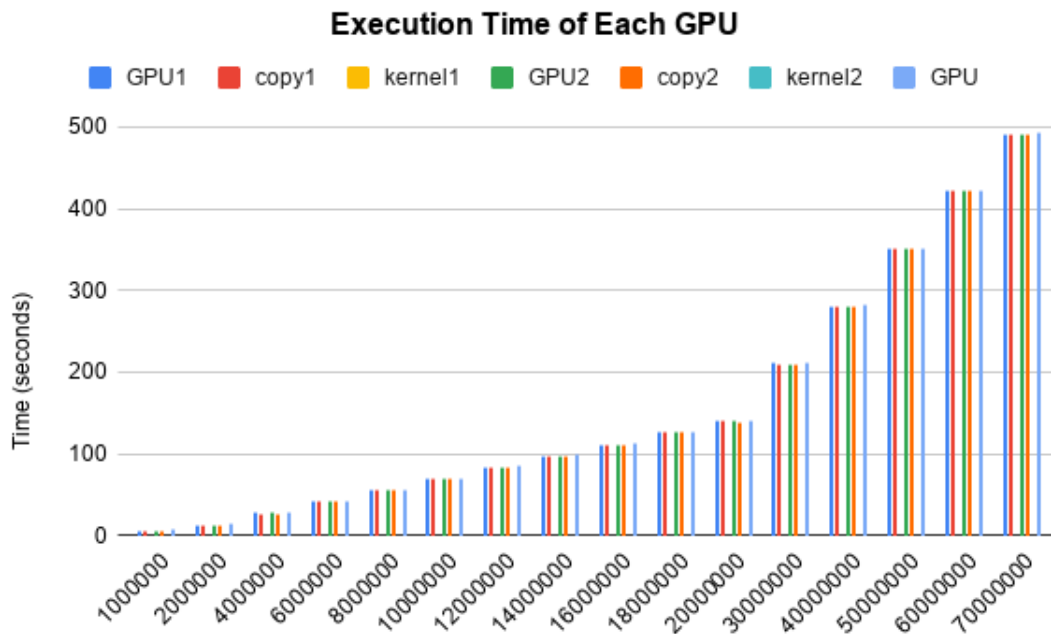


Figure 7.10 Execution time considering only multi-GPU execution, highlighting for each of them a) complete execution time, b) data transfer time, and c) kernel execution time, in an environment composed of two P100 GPUs.

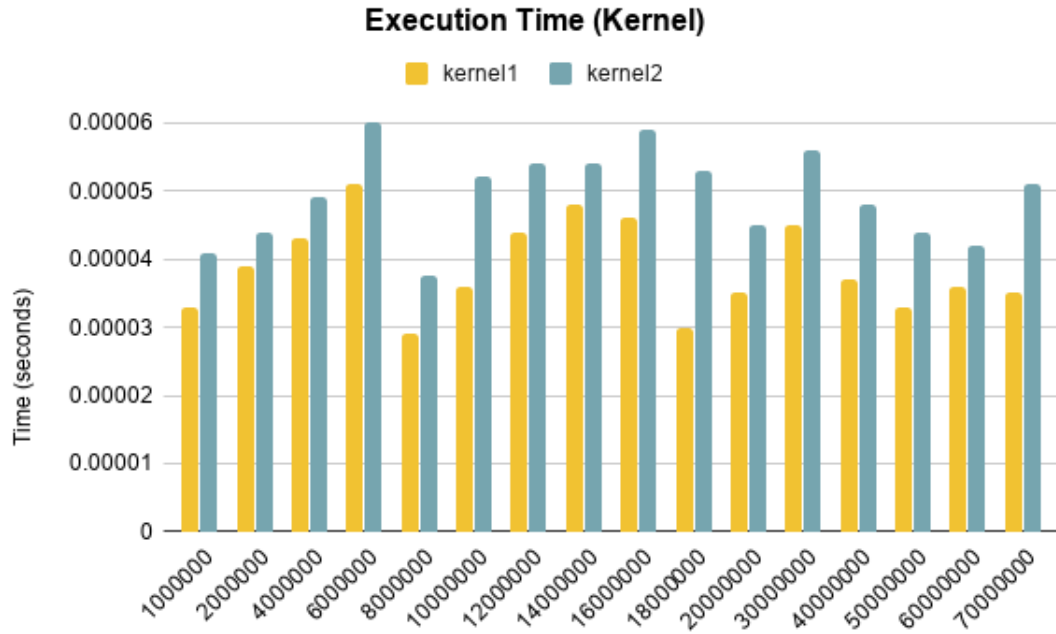


Figure 7.11 Kernel execution time of both GPUs in an environment composed of two P100 GPUs.

and device (dataset to be compared) and device and host (file containing the recovery IDs of the pair and their associated similarity index); and c) the time spent exclusively on the kernel that performs the probabilistic comparison. These metrics are repeated for the two GPUs involved in the process and the last bar in each set demonstrates the complete execution time for both GPUs. It is possible to observe that the execution time of each GPU is significantly influenced by the accumulated data transfer time (host to device and device to host). This finding reinforces the need to establish methods capable of minimizing the impact of data transfer time.

Figure 7.11 demonstrates more visually the execution time of the kernels executed by GPU 1 and GPU 2. Although the kernel executed in GPU 2 always has a slightly higher execution time, this fact will not be discussed, since both kernels execute at the same order of magnitude (10^{-5}).

The scalability tests presented here were performed considering only the load attributed to the CPU and using multicore parallel execution (40% of each input problem). The implemented solution employs the OpenMP paradigm to perform multithreading parallelism. The hardware used and described in the section 7.2.2.1 allows hyperthreading (MARR et al., 2002), that is why the system parameter concerning the number of threads varied from 1 to 80, with 1 being considered the serial time and 80 twice the amount of physical cores.

When performing strong scalability tests, the same input problem is applied to an increasing number of parallel threads. The purpose of this test was to analyze the performance gain of the application, given by the relationship between serial and parallel

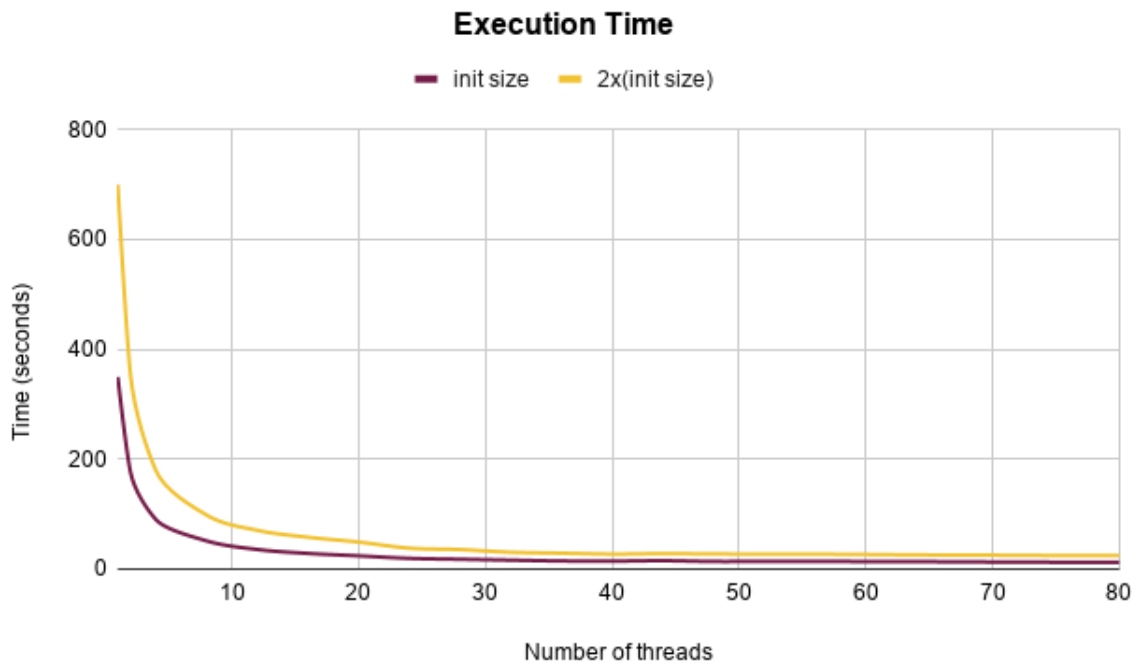


Figure 7.12 Strong scaling given by the execution time for two different input sizes in an environment composed of two P100 GPUs.

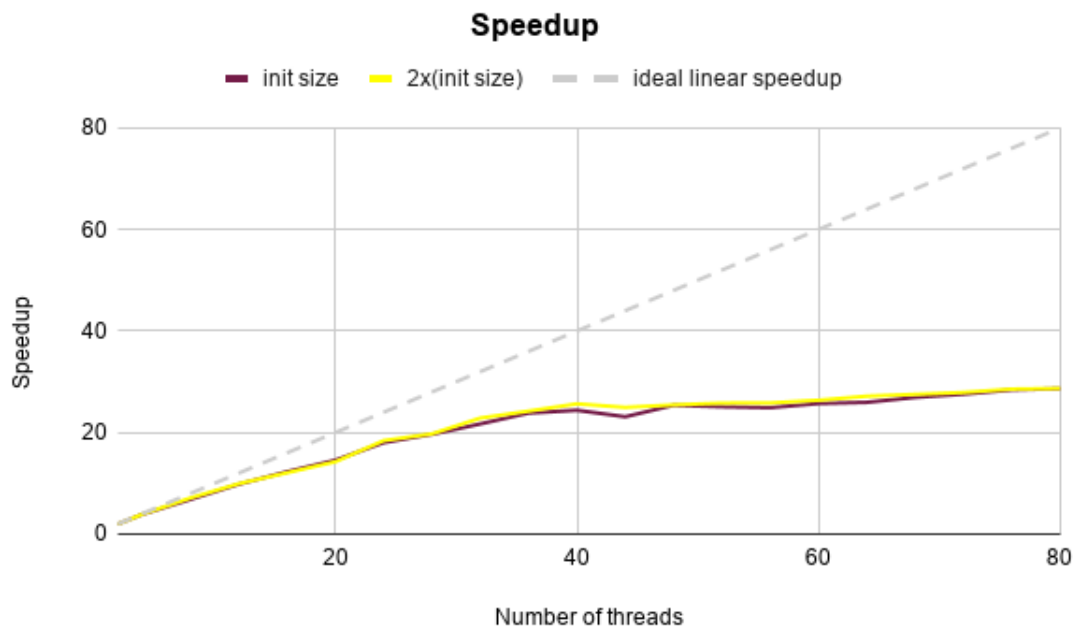


Figure 7.13 Strong scaling given by the speedup factor up to 80 threads for two different input sizes in an environment composed of two P100 GPUs.

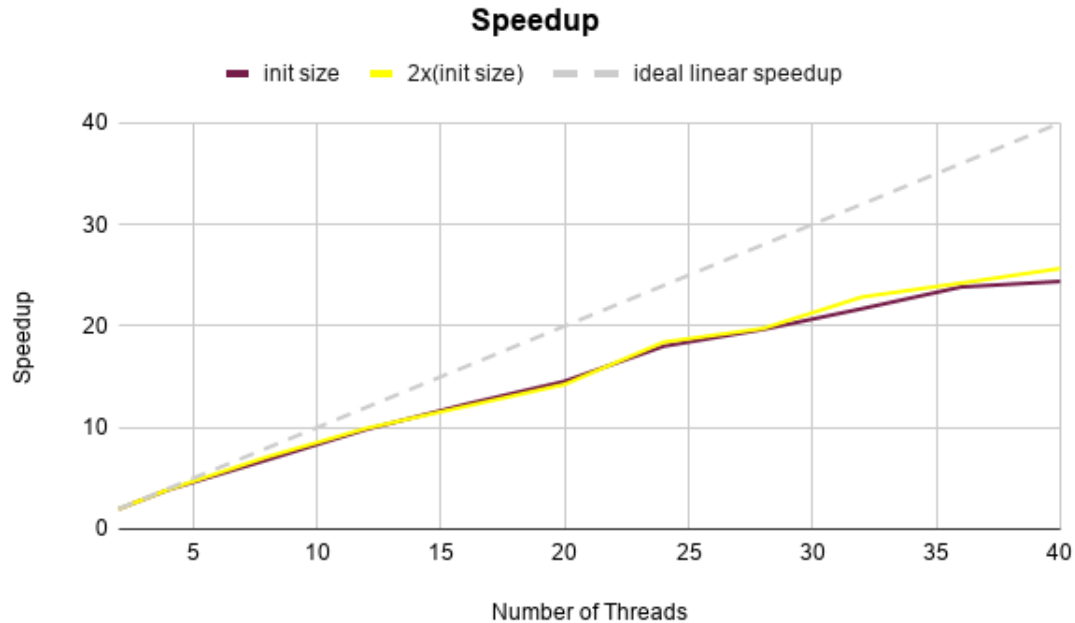


Figure 7.14 Strong scaling given by the speedup factor up to 40 threads for two different input sizes in an environment composed of two P100 GPUs.

execution (speedup factor), and the scalability limit, identifying when the gain stops being considerable. Through these tests, it is possible to observe the communication overhead and synchronization that are necessary to manage the load partition and the issues regarding the efficient use of shared resources. For the scalability tests demonstrated in this subsection, two input sizes were used. The first, described as "init size", refers to the 1 million records dataset and the second, described as "2x (init size)", refers to the 2 million records dataset.

Although the hardware used allows hyperthreading, the graph described in Figure 7.12 demonstrates that there is no performance gain when using this approach. The speedup graph described by Figure 7.13 and Figure 7.14, show more clearly that up to 15 threads, the application scales in a way compatible with the ideal speedup and between 15 and 40 threads there is still performance gain that compensates the parallelism.

In tests of weak scalability, the size of the input problem is increased in the same proportion of the increasing in the parallel resources (threads). Ideally, the use of shared resources should remain constant. In practice, however, the algorithms include extra steps of serialization, creation of threads, and communication that can occasionally increase execution time. The graph described in Figure 7.15 shows the results of weak scalability. The initial input problem has a size of 1 million records and is executed serially using 1 thread ("1x" on the horizontal axis). Each point on the horizontal axis considers twice the previous value for both parameters w and s . The ideal scalability curve, in this context, is a constant of the serial execution time. However, for higher problem size an overhead of w and s can be expected. Similar to the strong scalability tests, it is possible

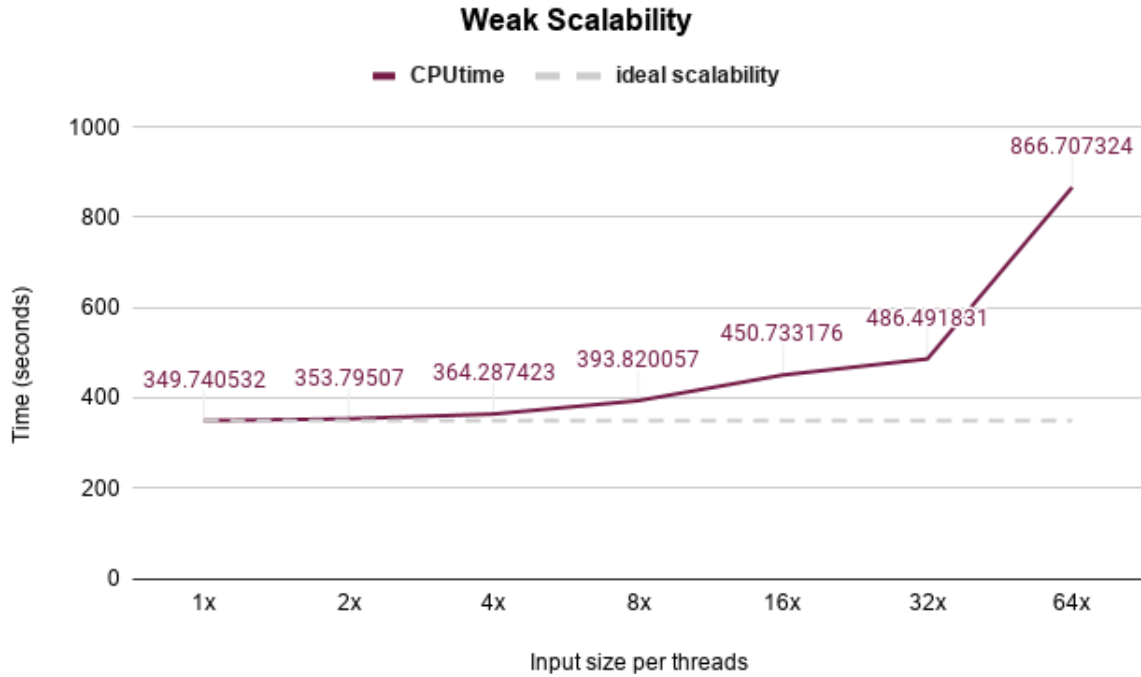


Figure 7.15 Weak scaling given by the execution time up to 64 threads and 64 million records in an environment composed of two P100 GPUs.

to observe that the use of hyperthreading is not useful to achieve performance gains.

7.2.3 Hybrid record linkage up to 100 million records

In this subsection, we describe the most recent tests of the GPU-accelerated hybrid probabilistic comparison that guarantees a comparison of 100 million records (larger database) without the use of blocking strategies. The evolution of the research focused on the optimization of data transfer, using NVLINK communication between devices to perform the deduplication step of the classified pairs for those blocks executed by the two GPUs. This incurs additional time in kernels execution but reduces time spent with data transfer for structures that store the similarity index of each classified pair, generating a positive impact on the total execution time of the application and optimizing the use of the PCI-e as a shared resource. The use of superior hardware was also decisive for storing a larger number of records in GPUs' global memory at once.

7.2.3.1 Experimental setup The tests presented in this subsection were performed on a GPU node consisting of two sockets Intel Xeon Gold 6148, with a frequency of 2.40GHz; 20 cores in each socket (representing a total of 40 cores in a single node); cache with 28MB; main memory equal to 192GB. The GPU node is composed of two Tesla V100-SXM2 graphic cards with global memory of 32GB, 900GB/s of memory bandwidth and 5120 *CUDA Cores*.

7.2.3.2 Performance analysis As shown in previous experiments, this subsection describes the performance evaluation results associated with the computing time of the comparison step from AtyImo-H pipeline when executed exclusively using the hybrid option. The number of OpenMP threads and the ratio threads per block are still used as system parameters (SP), and their amount is changed to observe scalability and performance issues. We varied the number of threads from 1 to 80. As application parameters (AP), we varied the input size defined by the number of records of the largest dataset. We varied the input size from 1 million to 100 million records. The smaller dataset still remained fixed in 10.000 records for all experiments. From 1 million to 20 million records, the sizes of the input problem were divided into regular intervals of 2 million records. From 20 million to 70 million the problem sizes were divided into regular intervals of 10 million records.

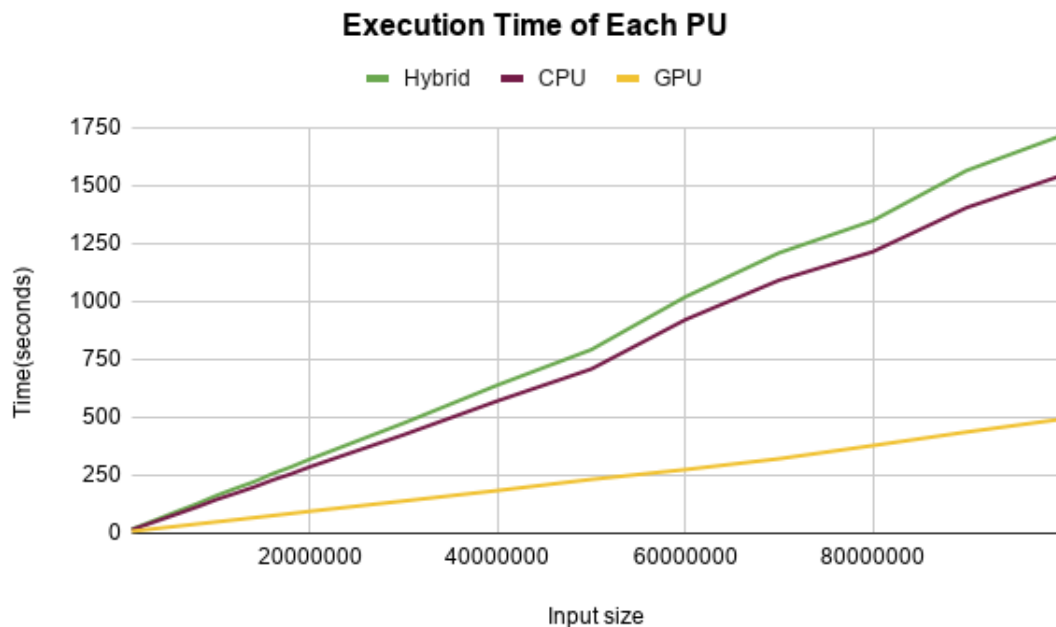


Figure 7.16 Execution time of hybrid approach and its comparison with CPU and GPU runtime in an environment composed of two V100 GPUs.

To analyze the execution time, as a start point, we varied the input problem size. Figure 7.16 shows the time of the hybrid approach (green) and the time spent with CPU execution (red) and GPU execution (yellow). The time spent in the block that performs loading was also considered in the total execution time and can be observed by Table 7.5. The multicore execution still limits the execution time of the hybrid approach and its isolated execution time is nearly the execution time for execution presented in subsection 7.2.2. This is due to the fact that the CPU characteristics for two computational environments are similar and the multicore algorithm had no significant changes. However, GPU execution time presented a significant execution time reduction and, besides that, it was possible to store the maximum number of records (100 million) required by AtyImo-H.

| | Load Time | CPU | GPU | Hybrid |
|-----------|------------------|-------------|------------|---------------|
| 1000000 | 2.236366 | 15.494517 | 8.949632 | 17.730507 |
| 2000000 | 3.428926 | 29.724551 | 13.418631 | 33.033304 |
| 4000000 | 5.547809 | 57.639354 | 22.389282 | 65.078163 |
| 6000000 | 7.105531 | 85.692782 | 31.308873 | 96.963078 |
| 8000000 | 9.804489 | 114.362579 | 40.328894 | 128.043082 |
| 10000000 | 12.859495 | 145.207225 | 49.337498 | 162.496411 |
| 12000000 | 15.785569 | 171.74601 | 58.263239 | 192.983175 |
| 14000000 | 18.349736 | 198.941357 | 67.335637 | 221.650761 |
| 16000000 | 21.293881 | 229.644019 | 76.402831 | 257.01596 |
| 18000000 | 24.5141 | 256.534511 | 85.512677 | 286.556365 |
| 20000000 | 37.506317 | 286.967554 | 95.066702 | 320.102923 |
| 30000000 | 49.592786 | 425.214275 | 139.498266 | 475.522523 |
| 40000000 | 58.844083 | 571.237995 | 184.521669 | 639.577269 |
| 50000000 | 71.902747 | 708.678306 | 232.654532 | 792.365382 |
| 60000000 | 83.054827 | 921.038331 | 274.735558 | 1019.378748 |
| 70000000 | 92.73407 | 1091.54025 | 321.594864 | 1209.430932 |
| 80000000 | 104.808594 | 1214.556377 | 378.462944 | 1348.359932 |
| 90000000 | 115.721562 | 1404.636898 | 436.912878 | 1564.787347 |
| 100000000 | 132.238598 | 1541.50828 | 491.608563 | 1713.792103 |

Table 7.5 Execution time of each PU and time spent loading datasets into devices' memory until 100 million records (time in seconds).

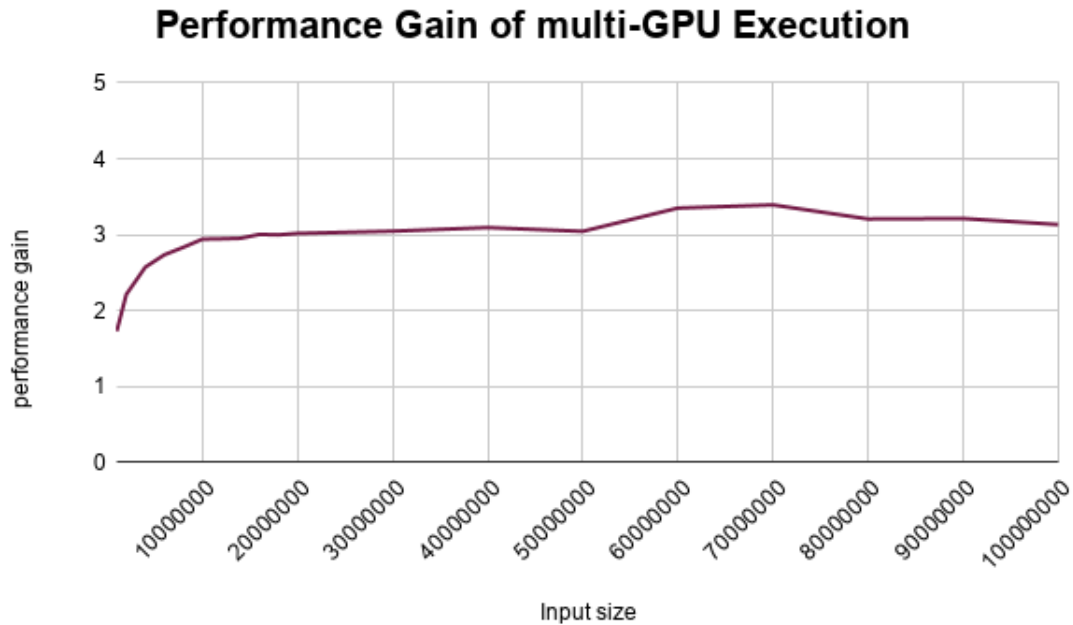


Figure 7.17 Performance gain of multi-GPU execution in comparison with CPU execution time in an environment composed of two P100 GPUs.

Figure 7.2.3.2 shows the performance gain for GPU execution when compared to CPU execution. The performance gain varies from 1.73x (small case with 1 million records) to 3.13x (largest case with 100 million records).

The graph described by Figure 7.18 shows the impact that each step has on the execution time: a) the time spent by the GPU; b) the time spent with transfer between host and device (dataset to be compared) and device and host (file containing the recovery IDs of the pair and their associated similarity index); and c) the time spent exclusively on the kernel that performs the probabilistic comparison. We can observe that the time that GPU 1 spends on data transfer is significantly reduced since only one GPU is sending data back to CPU. Figure 7.19 demonstrates in more details the execution time of both kernels.

To evaluate scalability for CPU execution we performed tests considering the runtime, the input size given by the number of records set to be compared and classified, the memory size required for each record set and the number of parallel threads. Similar to the computational environment presented in the previous section, the hardware used in this evaluation and described in the subsection 7.2.3.1 allows hyperthreading. Therefore, we varied the number of threads from 1 (serial execution) to 80 (maximum number of logical threads). Figure 7.20 shows a significant performance gain up to 20 threads and a slight performance gain up to 40 threads. From 40 to 80 threads, there is no performance improvement. The performance gain can be observed in more details in Figure 7.21 and Figure 7.22.

For the strong scalability tests demonstrated in this subsection, two input sizes were

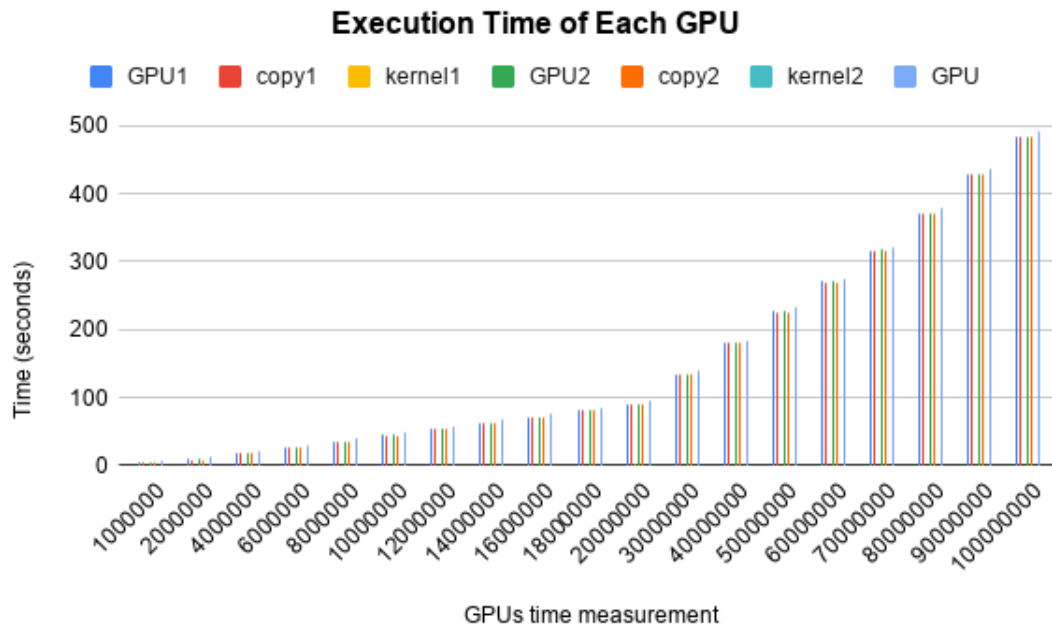


Figure 7.18 Execution time considering only multi-GPU execution, highlighting for each of them: a) complete execution time, b) data transfer time, and c) kernel execution time, in an environment composed of two V100 GPUs.

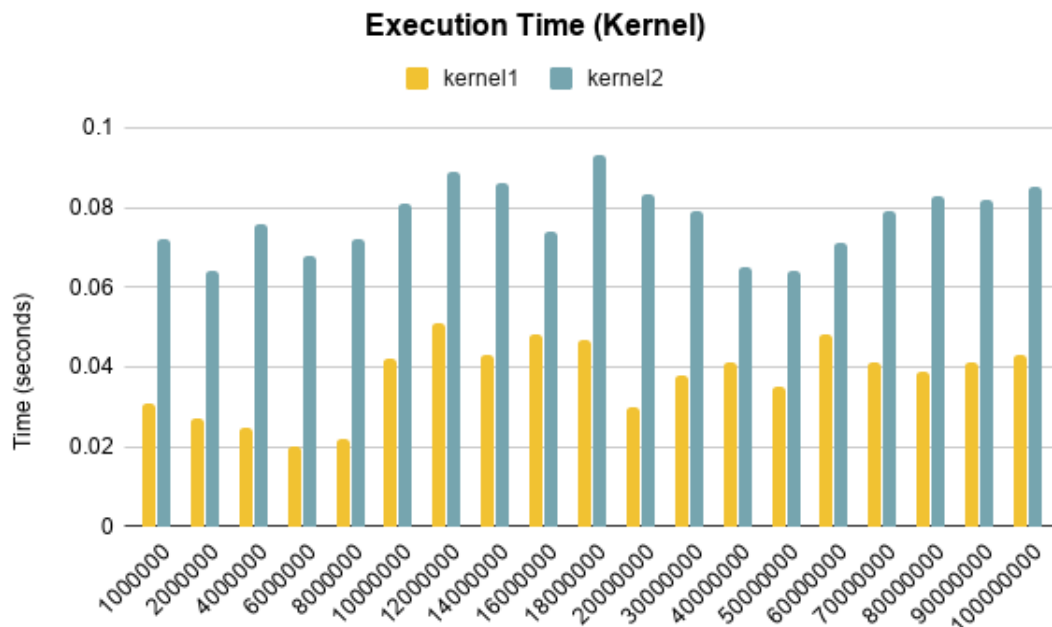


Figure 7.19 Kernel execution time of both GPUs in a environment composed of two V100 GPUs.

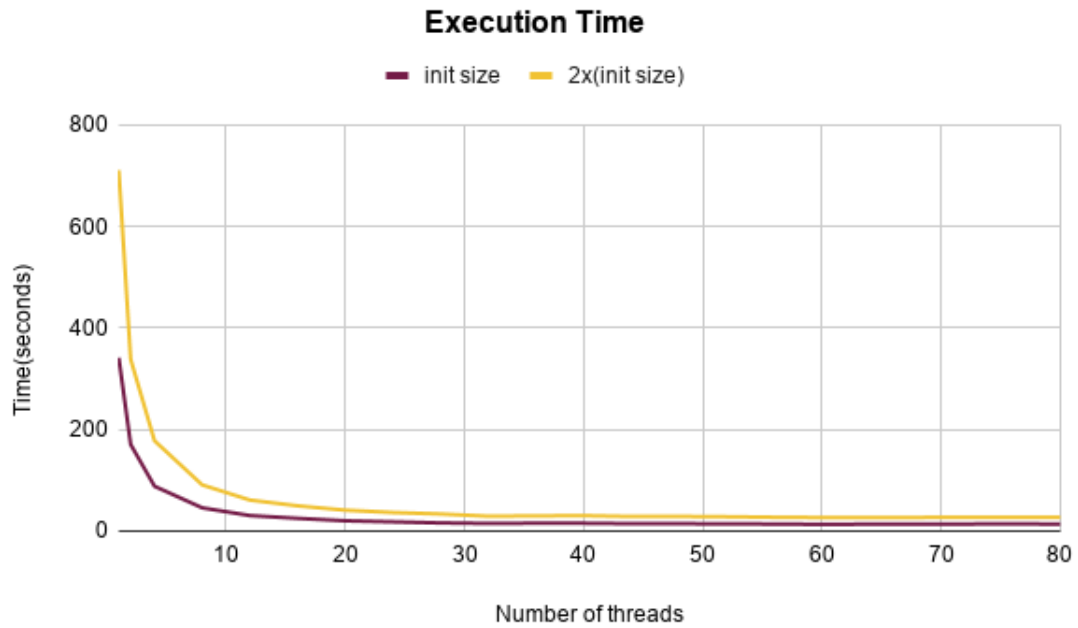


Figure 7.20 Strong scaling given by the execution time for two different input sizes in a environment composed of two V100 GPUs.

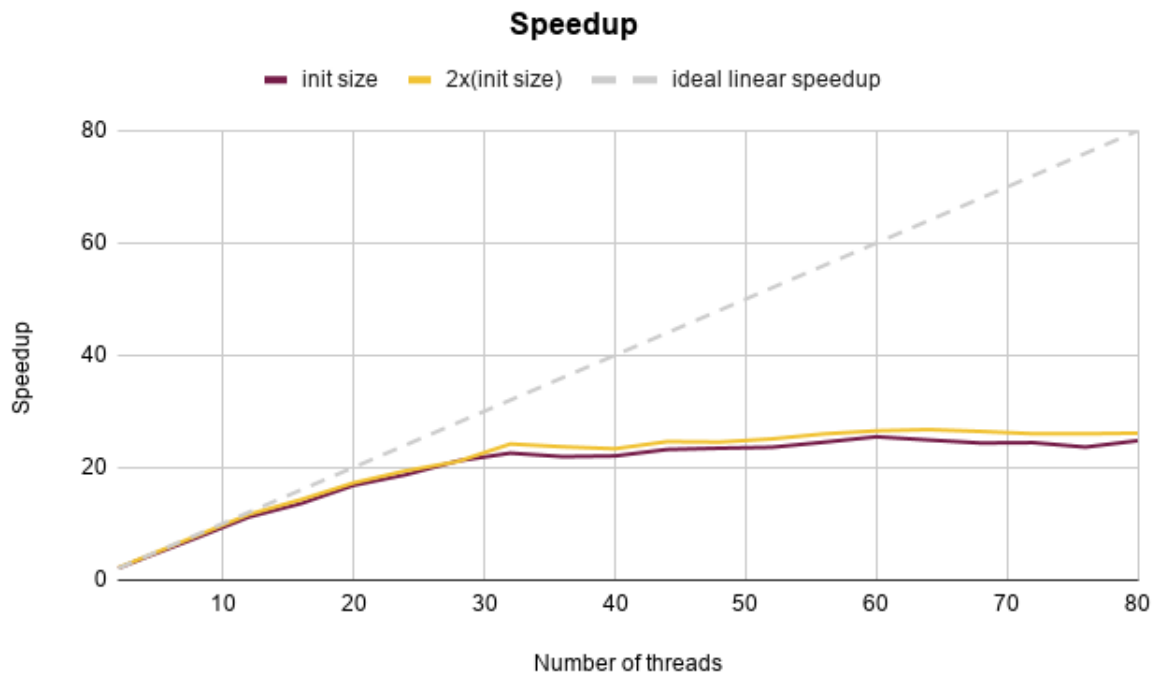


Figure 7.21 Strong scaling given by the speedup factor up to 80 threads for two different input sizes in a environment composed of two V100 GPUs.

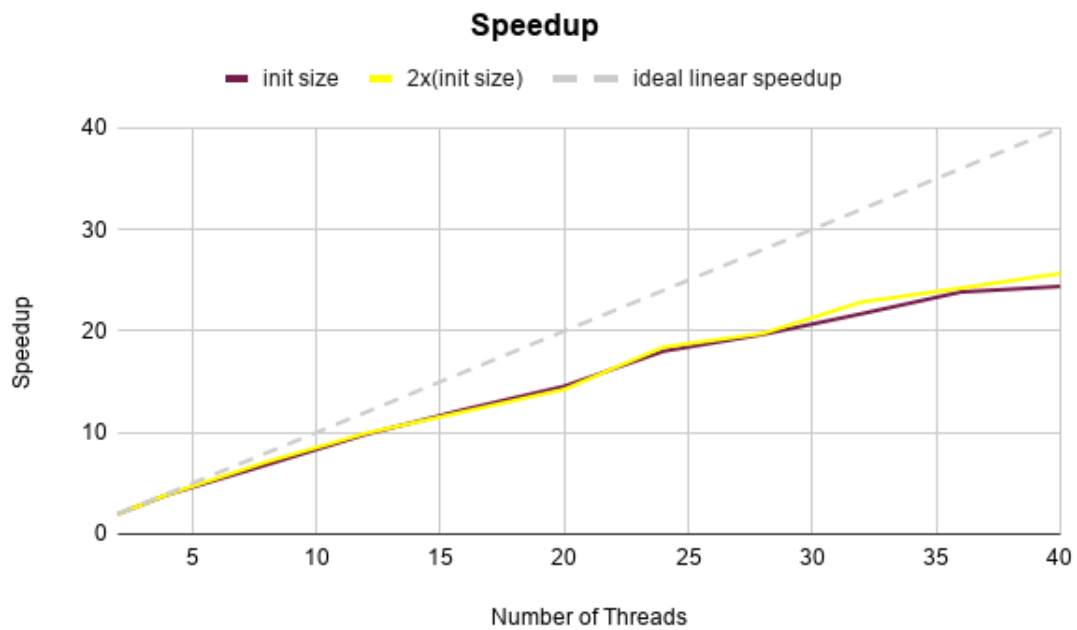


Figure 7.22 Strong scaling given by the speedup factor up to 40 threads for two different input sizes in a environment composed of two V100 GPUs.

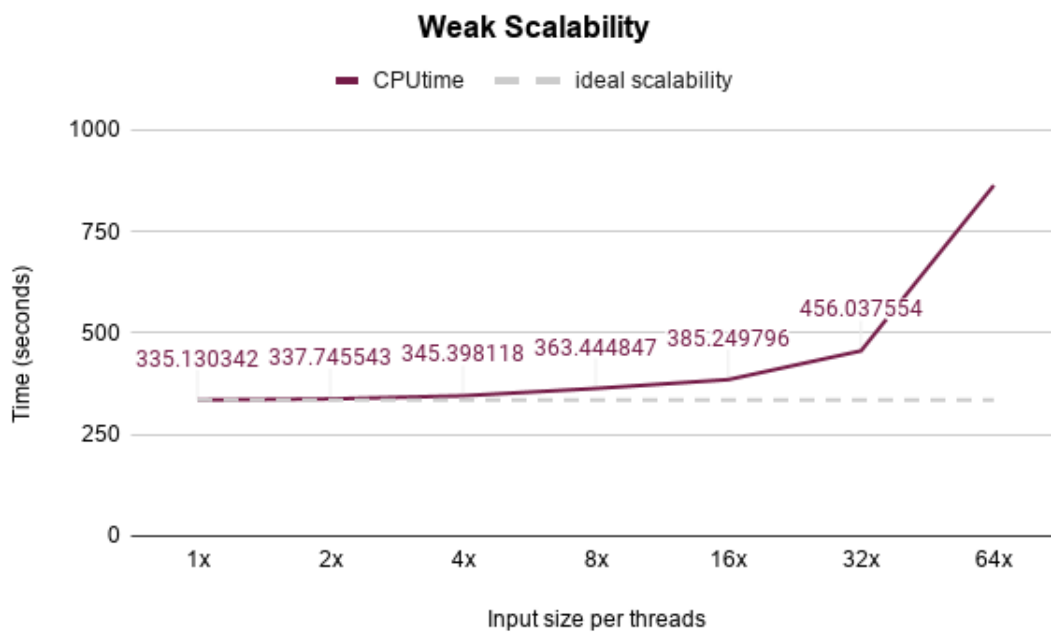


Figure 7.23 Weak scaling given by the execution time up to 64 threads and 64 million records in a environment composed of two V100 GPUs.

| | data size | data size CPU | data size GPU1 | data size GPU2 |
|------------------|-----------|------------------|-------------------|-------------------|
| 1000000 | 0.424 | 0.1696 | 0.1272 | 0.1272 |
| 2000000 | 0.848 | 0.3392 | 0.2544 | 0.2544 |
| 4000000 | 1.696 | 0.6784 | 0.5088 | 0.5088 |
| 6000000 | 2.544 | 1.0176 | 0.7632 | 0.7632 |
| 8000000 | 3.392 | 1.3568 | 1.0176 | 1.0176 |
| 10000000 | 4.24 | 1.696 | 1.272 | 1.272 |
| 12000000 | 5.088 | 2.0352 | 1.5264 | 1.5264 |
| 14000000 | 5.936 | 2.3744 | 1.7808 | 1.7808 |
| 16000000 | 6.784 | 2.7136 | 2.0352 | 2.0352 |
| 18000000 | 7.632 | 3.0528 | 2.2896 | 2.2896 |
| 20000000 | 8.48 | 3.392 | 2.544 | 2.544 |
| 30000000 | 12.72 | 5.088 | 3.816 | 3.816 |
| 40000000 | 16.96 | 6.784 | 5.088 | 5.088 |
| 50000000 | 21.2 | 8.48 | 6.36 | 6.36 |
| 60000000 | 25.44 | 10.176 | 7.632 | 7.632 |
| 70000000 | 29.68 | 11.872 | 8.904 | 8.904 |
| 80000000 | 33.92 | 13.568 | 10.176 | 10.176 |
| 90000000 | 38.16 | 15.264 | 11.448 | 11.448 |
| 100000000 | 42.4 | 16.96 | 12.72 | 12.72 |

Table 7.6 Input data sizes loaded into each Processing Unit for different numbers of records.

used following the same structure described in section 7.2.2.2 (1 million records and 2 million records).

For weak scalability tests, the input size grows at the same proportion as the number of parallel threads grows. Figure 7.23 demonstrates a satisfactory performance gain up to 32x (horizontal axis) using 32 threads and a dataset with 32 million records, reaffirming that the use of hyperthreading is not advantageous at any stage of the parallel execution. The scalability measures presented here are useful to indicate the ability of AtyImo-H (GPU) to deliver greater computational power when the amount of resources is increased.

AtyImo-H (GPU) is a GPU-accelerated multicore system controlled by host threads executed on dedicated CPU cores. The workload is partitioned between host and devices (CPU and GPUs). The devices have a separate memory, therefore, it is needed to transfer input data to the devices and then transfer resulting data back to the host. Table 7.6 shows the input data size loaded into each Processing Unit for different numbers of records.

7.2.4 Comparison with Spark-based AtyImo

The presentation of AtyImo-H as a hybrid tool seeks to offer a scalable and high-performance alternative to meet the requirement for timely responses, which is one of the main requirements for the employment of this type of application. It is necessary

to emphasize, however, that the Spark-based AtyImo pipeline, built with a technology that explores iterative algorithms and in-memory computation, benefits the application in different requirements such as failure management, data recovery, and maintenance.

The data replication management offered by Spark has proven performance and benefits several applications that need to scale on different nodes of a distributed cluster composed of CPU processing units. Spark's superior performance compared to other MapReduce jobs has already been punctuated in several works since this tool keeps the data in memory, avoiding reloading when searching for data from disk in intermediate steps, as done by other distributed frameworks like Hadoop. Therefore, this solution must be taken into account when developing generic big data applications.

Nevertheless, solutions based on a native implementation offer consistent performance and transcend Spark-based solutions concerning processing speed (REYES-ORTIZ; ONETO; ANGUIA, 2015). If, on the one hand, such solutions are capable of offering superior performance, on the other, they lack fault tolerance, as already mentioned, requiring a cautious use of distributed message passing approaches.

The heterogeneity of execution platforms and the current architecture diversity are essential factors to justify the exploration of multiple levels of parallelism, to extract maximum performance from the resources used, since the execution time is an indispensable requirement for most applications involving big data analysis, such as record linkage, mainly in the pipeline stage dedicated to probabilistic comparison. The exploration of the heterogeneous platform also provides higher throughput, allowing a single computing node to process a large volume of data in less time, reducing the need for process distribution.

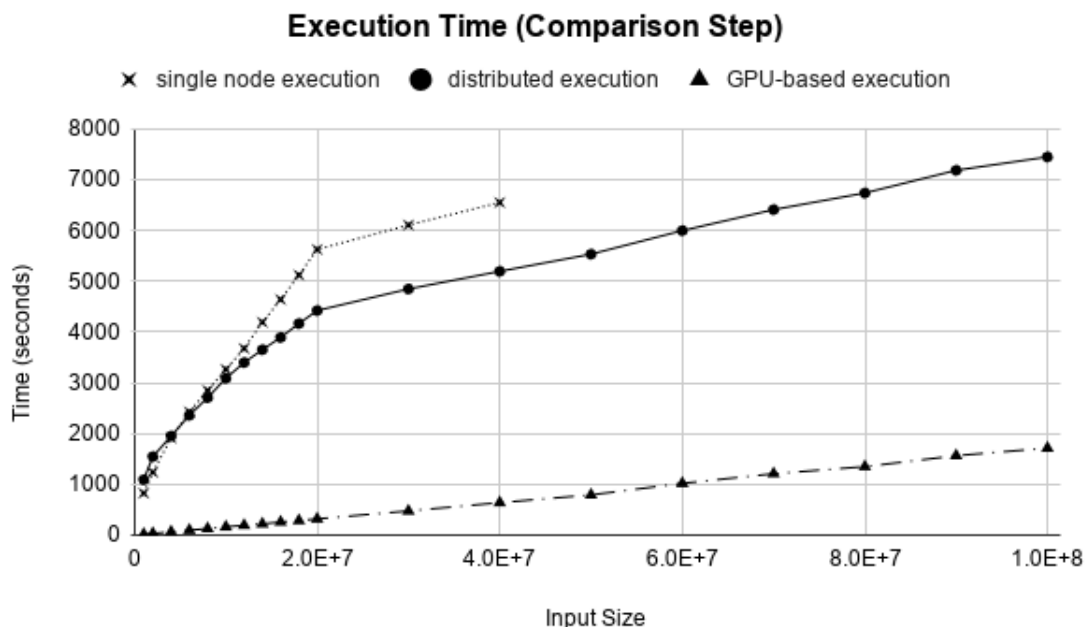


Figure 7.24 AtyImo-H CPU X GPU-based execution (pairwise comparison step only).

Figure 7.24 establishes a performance comparison between AtyImo-H and the Spark-based AtyImo. It is possible to observe that the hybrid execution (triangle line) surpasses the Spark-based solution in a meaningful manner. We test Spark-based in a single-node environment (x- line) to provide a fair comparison with a single node AtyImo-H execution. Due to this scenario, it was possible to perform a probabilistic comparison with an input size of up to 40 million records (dataset equivalent to 16.96GB) due to the framework's memory limitation. When compared to single-node execution, AtyImo-H's performance ranged from 46.5x to 10.2x when the input size ranged from 0.424GB (smaller dataset) to 16.96 GB (larger dataset). The Spark-based execution was also submitted in a distributed environment (dots line) using 40 cores per node and 10 computational nodes. In this execution, it is possible to observe that the application performance improved as the input data volume increased. When compared to distributed execution, AtyImo-H's performance ranged from 61x to 4x when the input size ranged from 0.424GB (smaller dataset) to 42.4GB (larger dataset). The variation of the execution time in different input sizes reaffirms the distributed framework's effectiveness to process huge volumes of data. When the volume of data increases, the speedup for high performance execution tend to be smaller. The graph highlights the feasibility of AtyImo-H as a hybrid tool relying on different programming paradigms and processing units to offer high-performance for big data applications. Thus, it is possible to adapt the application parameters to the execution infrastructure available in that specific scenario.

This chapter presents a summary of the goals, development and results obtained by the execution of this work. Subsequent sections present the challenges encountered and the strategies to overcome them. Future work is also presented.

CONCLUSION

8.1 FINAL CONSIDERATIONS

The objective of this work was to present technological alternatives to enable the implementation and execution of the module that performs the probabilistic comparison of records contextualized in a Big Data scenario, in heterogeneous environments composed by multicore (CPU) and manycore (GPU) processing units. It was also intended to integrate the developed solutions and apply the considered studies to the *AtyImo* (PINTO et al., 2017c) tool pipeline, previously presented and partially developed during the development of this work. This integration allows researchers already using *AtyImo* to speed up the execution of the linkage process over large datasets whenever heterogeneous hardware is available.

One of the main advantages of frameworks like Spark is related to fault tolerance. But, in HPC clusters dedicated to scientific research, the high-quality of high-end hardware ensures that failures are not a problem for information security and jobs execution. On the other hand, security must be ensured by physical infrastructure (redundancy etc.), as well as through a certain level of control in the execution environment (queues, checkpoint etc.). Due to that, fault tolerance is not presented as a drawback for the proposed solution.

The scope of this work is the probabilistic processing of structured datasets similar to those generated by government information systems managed by the Ministry of Health and the Ministry of Development. In this research, we used as proof of concept, datasets of the Hospital Information System (SIH), the Mortality Information System (SIM), the Notification Information System (SINAN), the Live Births Information System (SINASC), the Unified Registry for Social Programmes (Cadastro Unico) and the Bolsa Família Program (PBF). The developed solutions were applied over real bases (for proof of concept and accuracy analysis), as well as over synthetic bases (for scalability analysis and performance matters). Chapter 7, for example, presents results of a linkage

between CADU and SIM to gather socioeconomic information and mortality data from Sergipe (SE), Santa Catarina (SC) and Rondônia (RO) state registries. Considering that data obtained from government-based information systems vary regionally in quality, it is important to select data from different regions to ensure that the obtained accuracy results represent the quality of the method for all cases.

Still in Chapter 7, different aspects of the hybrid application behavior (CPU and GPU acting collaboratively) were demonstrated for the comparison step that performs the probabilistic linkage of records. First, a preliminary approach was shown through which it is possible to compare and classify up to 20 million records. An initial calibration fixed a static workload division, which achieved optimum performance in the test environment, showing results superior to *multicore-only* execution and *manycore-only*. For replicating the experiments in environments with a different configuration, however, a new calibration (installation) step was required.

The applied static division distributed 70% of the workload to GPUs (35% for each GPU) and the remaining to the *host* for CPU processing. Other distributions were also tested (40% for each GPUs and 20% for one CPU and 45% for each GPU and 10% for the CPU), however, the results did not exceed the results presented. So far, the definition of the workload division between the involved PUs is decided in a calibration step performed before the executions themselves. Through the history of calibration runs, one can determine the initial parameters. The disadvantage of this approach is that for each new execution environment (different hardware) a new calibration is required to verify the results obtained for the performance metrics considered.

For runs that scaled up to 70 million records, the ratio of 35% for each GPU and 30% for the CPU was maintained. For these tests, an execution environment consisting of two GPUs and one CPU node was also maintained. However, the hardware configuration and software versions have been updated. Tesla K80 graphics cards were replaced by two Tesla P100-SXM22 cards interconnected via NVLINK. The node consisting of four Intel Xeon CPUs with 2.93 GHz and 40 cores was replaced by a node consisting of two Intel Xeon Gold 6148 CPUs, 2.40GHz with 20 cores each, also representing a total of 40 cores. DDR3 main memory capacity of 130 GB was replaced by one with capacity of 196 GB. The CUDA version used has been updated from 4.0 to 9.0.

Finally, developments in parallel implementation and improvements in the comparison and synchronization function between devices have allowed 100 million records to be paired without any blocking or indexing method in less than an hour. To the best of our knowledge, no tool was found in the literature capable of linking this amount of records with the performance offered by AtyImo-H.

The results presented demonstrate that the hybrid approach has superior performance in comparison with executions on a CPU or GPU exclusively. It is clear from tests that a significant reduction in runtime is sufficient to compensate for the extra time spent with data transfer, synchronization, and communication between PUs. The results section also demonstrated the superiority of AtyImo-H when compared to Spark-based AtyImo in terms of execution time.

The AtyImo-H tool is available in a public repository¹ along with test data and user-guide and aims to meet different demands for integrating structured datasets. Although it has been calibrated to handle datasets from specific government domains, AtyImo-H can be customizable to suit different sources. Since the application execution is grouped in well-defined blocks and present a pipeline-based design, it is even easier to use and integrate its functionalities.

8.2 FUTURE WORK

This work intends to cooperate for research projects that require probabilistic record linkage in large government datasets (or similar).

Future developments can involve:

- fully validate the proposed approach using other linkage scenarios, to check for accuracy and performance;
- providing an integrated, fully customizable implementation of AtyImo-H allowing the user to easily define application parameters (AP) that can influence on the linkage methods and workload distribution calibration;
- linking multiple (> 2) datasources simultaneously, through the coordinated use of multiple GPUs and a bespoke workload balancing strategy;
- optimize memory management by exploring CUDA unified memory and zero-copy memory as alternative data transfer modality.

¹<https://github.com/atyimo-lab/atyimo-h>

BIBLIOGRAPHY

- ALKATHERI, S.; ABBAS, S.; SIDDIQUI, M. A comparative study of big data frameworks. *International Journal of Computer Science and Information Security (IJCSIS)*, v. 17, n. 1, 2019.
- ALMEIDA, B. de A. et al. The center for data and knowledge integration for health (cidacs). *International Journal of Population Data Science*, v. 4, n. 2, 2019.
- ALOWAYYED, S. et al. Multiscale computing in the exascale era. *Journal of Computational Science*, Elsevier, v. 22, p. 15–25, 2017.
- AMARAL, F. *Introdução à Ciência de Dados: mineração de dados e big data*. [S.l.]: Alta Books Editora, 2016.
- AMD. *AMD Ryzen™ Threadripper™ 3970X Processor*. 2019. Disponível em: <https://www.amd.com/en/products/cpu/amd-ryzen-threadripper-3970x/>.
- ANDERSON, M. et al. Bridging the gap between hpc and big data frameworks. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 10, n. 8, p. 901–912, 2017.
- BARBOSA, G. C. et al. CIDACS-RL: A novel search engine-based record linkage system for huge datasets with high accuracy and scalability. In: WILEY 111 RIVER ST, HOBOKEN 07030-5774, NJ USA. *Pharmacoepidemiology and Drug Safety*. [S.l.], 2019. v. 28, p. 118–118.
- BARRETO, M. E. et al. Assessing the accuracy of probabilistic record linkage of social and health databases in the 100 million brazilian cohort. *International Journal of Population Data Science*, v. 1, n. 1, 2017.
- BERGMAN, K. et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, v. 15, 2008.
- BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, v. 13, n. 7, p. 422–426, 1970.
- BOLZ, J. et al. Sparse matrix solvers on the gpu: conjugate gradients and multigrid. v. 22, n. 3, p. 917–924, 2003.
- BORATTO, M. et al. Exploring hybrid parallel systems for probabilistic record linkage. *The Journal of Supercomputing*, Springer, p. 1–13, 2018.

- CARBONE, P. et al. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, IEEE Computer Society, v. 36, n. 4, 2015.
- CARTER, D. J. et al. The impact of a cash transfer programme on tuberculosis treatment success rate: a quasi-experimental study in brazil. *BMJ global health*, BMJ Specialist Journals, v. 4, n. 1, p. e001029, 2019.
- CATANZARO, B.; SUNDARAM, N.; KEUTZER, K. Fast support vector machine training and classification on graphics processors. In: ACM. *Proceedings of the 25th international conference on Machine learning*. [S.l.], 2008. p. 104–111.
- CHEN, Y. et al. Dadianna: A machine-learning supercomputer. In: IEEE COMPUTER SOCIETY. *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. [S.l.], 2014. p. 609–622.
- CHRISTEN, P. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE transactions on knowledge and data engineering*, IEEE, v. 24, n. 9, p. 1537–1555, 2011.
- CIDACS. *Coorte de 100 Milhões de Brasileiros*. 2015. Disponível em: <https://cidacs.bahia.fiocruz.br/plataforma/coorte-de-100-milhoes-de-brasileiros/>.
- CLARKE, D.; LASTOVETSKY, A.; RYCHKOV, V. Column-based matrix partitioning for parallel matrix multiplication on heterogeneous processors based on functional performance models. In: SPRINGER. *European Conference on Parallel Processing*. [S.l.], 2011. p. 450–459.
- CLARKE, D. et al. Fupermod: A framework for optimal data partitioning for parallel scientific applications on dedicated heterogeneous hpc platforms. In: SPRINGER. *International Conference on Parallel Computing Technologies*. [S.l.], 2013. p. 182–196.
- COX, M.; ELLSWORTH, D. Application-controlled demand paging for out-of-core visualization. In: IEEE. *Proceedings. Visualization'97 (Cat. No. 97CB36155)*. [S.l.], 1997. p. 235–244.
- DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, ACM, v. 51, n. 1, p. 107–113, 2008.
- DOAN, A.; HALEVY, A.; IVES, Z. Principles of data integration. Elsevier Science, 2012. Disponível em: <http://books.google.com.br/books?id=s2YCKGrO10YC>.
- DUARTE, J. M.; SANTOS, J. B. d.; MELO, L. C. Comparison of similarity coefficients based on rapid markers in the common bean. *Genetics and Molecular Biology*, SciELO Brasil, v. 22, n. 3, p. 427–432, 1999.
- ELFEKY, M. G.; VERYKIOS, V. S.; ELMAGARMID, A. K. Tailor: A record linkage toolbox. In: IEEE. *Proceedings 18th International Conference on Data Engineering*. [S.l.], 2002. p. 17–28.

- ÉTIENNE, E. Y. *Hyper-threading*. [S.l.]: TurbsPublishing, 2012.
- FELLEGI, I. P.; SUNTER, A. B. A theory for record linkage. *Journal of the American Statistical Association*, v. 64, p. 1183–1210, 1969.
- FORCHHAMMER, B. et al. Duplicate detection on gpus. *HPI Future SOC Lab*, v. 70, n. 3, 2013.
- FRANKE, M. et al. Scads research on scalable privacy-preserving record linkage. *Datenbank-Spektrum*, Springer, v. 19, n. 1, p. 31–40, 2019.
- FUNG, J.; TANG, F.; MANN, S. Mediated reality using computer graphics hardware for computer vision. In: IEEE. *Wearable Computers, 2002.(ISWC 2002). Proceedings. Sixth International Symposium on*. [S.l.], 2002. p. 83–89.
- GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The google file system. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. [S.l.: s.n.], 2003. p. 29–43.
- GREGG, C.; HAZELWOOD, K. Where is the data? why you cannot debate cpu vs. gpu performance without the answer. In: IEEE. *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*. [S.l.], 2011. p. 134–144.
- GSCHWIND, T. et al. Fast record linkage for company entities. In: IEEE. *2019 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2019. p. 623–630.
- HALL, R.; FIENBERG, S. E. Privacy-preserving record linkage. In: SPRINGER. *International conference on privacy in statistical databases*. [S.l.], 2010. p. 269–283.
- HAWKINS, S. S. et al. The linked century study: linking three decades of clinical and public health data to examine disparities in childhood obesity. *BMC pediatrics*, Springer, v. 16, n. 1, p. 32, 2016.
- HIVE, A. Apache hive. 2013.
- HOLMAN, C. D. J. et al. A decade of data linkage in western australia: strategic design, applications and benefits of the wa data linkage system. *Australian Health Review*, CSIRO, v. 32, n. 4, p. 766–777, 2008.
- INTEL. *NVIDIA Ampere Architecture In-Depth*. 2019. Disponível em: <https://www.intel.com/content/www/us/en/products/programmable/fpga/stratix-10/gx.html/>.
- ISLAM, N. S. et al. Accelerating i/o performance of big data analytics on hpc clusters through rdma-based key-value store. In: IEEE. *2015 44th International Conference on Parallel Processing*. [S.l.], 2015. p. 280–289.
- KAY, D.; HARMELEN, M. v. Activity data: Delivering benefits from the data deluge. *Accessed: Jan*, 2014.

KIM, H.-s.; LEE, D. Parallel linkage. In: ACM. *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. [S.l.], 2007. p. 283–292.

KÖPCKE, H.; RAHM, E. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, Elsevier, v. 69, n. 2, p. 197–210, 2010.

KRASHINSKY, R. et al. *NVIDIA Ampere Architecture In-Depth*.

LANEY, D. 3d data management: Controlling data volume, velocity and variety. *META group research note*, v. 6, n. 70, p. 1, 2001.

LEMIEUX, V. L.; GORMLY, B.; ROWLEDGE, L. Meeting big data challenges with visual analytics: The role of records management. *Records Management Journal*, Emerald Group Publishing Limited, v. 24, n. 2, p. 122–141, 2014.

LI, Z. et al. Ipso: A scaling model for data-intensive applications. In: IEEE. *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. [S.l.], 2019. p. 238–248.

LIVINGSTONE, S. J. et al. Estimated life expectancy in a scottish cohort with type 1 diabetes, 2008-2010. *Jama*, American Medical Association, v. 313, n. 1, p. 37–44, 2015.

LUCENE, A. Apache lucene-overview. *Internet: <http://lucene.apache.org/java/docs/>*[Jan. 15, 2009], 2010.

MACHADO, D. B. et al. Conditional cash transfer programme: Impact on homicide rates and hospitalisations from violence in brazil. *PloS one*, Public Library of Science, v. 13, n. 12, p. e0208925, 2018.

MAHESWARAN, M. et al. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. 1999.

MARR, D. T. et al. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, v. 6, n. 1, 2002.

MCCUNE, B.; GRACE, J.; URBAN, D. Analysis of ecological communities. MJM Software Design, 2002.

MESTRE, D. G.; PIRES, C. E.; NASCIMENTO, D. C. Adaptive sorted neighborhood blocking for entity matching with mapreduce. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. [S.l.: s.n.], 2015. p. 981–987.

MITTAL, S.; VETTER, J. S. A survey of cpu-gpu heterogeneous computing techniques. *ACM Computing Surveys (CSUR)*, ACM, v. 47, n. 4, p. 69, 2015.

OLIVEIRA, M. A. P. de; PARENTE, R. C. M. Cohort and case-control studies in the evidence-based medicine era. 2010.

PINNECKE, M.; BRONESKE, D.; SAAKE, G. Toward gpu accelerated data stream processing. In: *GvD*. [S.l.: s.n.], 2015. p. 78–83.

PINTO, C.; BARRETO, M.; BORATTO, M. Auto-tuning trsm with an asynchronous task assignment model on multicore, multi-gpu and coprocessor systems. In: IEEE. *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*. [S.l.], 2016. p. 1–8.

PINTO, C. et al. Scaling probabilistic record linkage on multicore and multi-gpu system. *17th International Conference on Computational and Mathematical Methods in Science and Engineering*, CMMSE, 2017.

PINTO, C. et al. Accuracy of probabilistic record linkage applied to the brazilian 100 million cohort project. In: IEEE ENGINEERING IN MEDICINE AND BIOLOGY SOCIETY (EMBS). [S.l.], 2017.

PINTO, C. et al. Probabilistic integration of large brazilian socioeconomic and clinical databases. In: IEEE. *2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS)*. [S.l.], 2017. p. 515–520.

PITA, R. et al. A machine learning trainable model to assess the accuracy of probabilistic record linkage. In: SPRINGER. *Big Data Analytics and Knowledge Discovery – DaWak 2017*. [S.l.], 2017. p. 214–227.

PITA, R. et al. A spark-based workflow for probabilistic record linkage of healthcare data. p. 17–26, 2015.

PITA, R. et al. On the accuracy and scalability of probabilistic data linkage over the brazilian 114 million cohort. *IEEE journal of biomedical and health informatics*, 2018.

RASCH, A. et al. High-performance probabilistic record linkage via multi-dimensional homomorphisms. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. [S.l.: s.n.], 2019. p. 526–533.

REYES-ORTIZ, J. L.; ONETO, L.; ANGUIA, D. Big data analytics in the cloud: Spark on hadoop vs mpi/openmp on beowulf. In: *INNS Conference on Big Data*. [S.l.: s.n.], 2015. v. 8, p. 121.

RISTAD, E. S.; YIANILOS, P. N. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 20, n. 5, p. 522–532, 1998.

SCHALLER, R. R. Moore’s law: past, present and future. *IEEE spectrum*, IEEE, v. 34, n. 6, p. 52–59, 1997.

SEHILI, Z. et al. Privacy preserving record linkage with ppjoin. *Datenbanksysteme für Business, Technologie und Web (BTW 2015)*, Gesellschaft für Informatik eV, 2015.

SHVACHKO, K. et al. The hadoop distributed file system. In: IEEE. *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. [S.l.], 2010. p. 1–10.

- SINGH, D.; REDDY, C. K. A survey on platforms for big data analytics. *Journal of Big Data*, Nature Publishing Group, v. 2, n. 1, p. 8, 2015.
- SUGGS, D.; SUBRAMONY, M.; BOUVIER, D. The amd “zen 2” processor. *IEEE Micro*, IEEE, v. 40, n. 2, p. 45–52, 2020.
- TUKEY, J. W. *Exploratory Data Analysis: Limited Preliminary Ed.* [S.l.]: Addison-Wesley Publishing Company, 1970.
- VATSALAN, D.; CHRISTEN, P.; RAHM, E. Incremental clustering techniques for multi-party privacy-preserving record linkage. *Data & Knowledge Engineering*, Elsevier, v. 128, p. 101809, 2020.
- WANG, B.; SCHMIDL, D.; MÜLLER, M. S. Evaluating the energy consumption of openmp applications on haswell processors. In: SPRINGER. *International Workshop on OpenMP*. [S.l.], 2015. p. 233–246.
- WENDEL, D. et al. The power7 tm processor soc. In: IEEE. *IC Design and Technology (ICICDT), 2010 IEEE International Conference on.* [S.l.], 2010. p. 71–73.
- ZAHARIA, M. et al. Spark: Cluster computing with working sets. *HotCloud*, v. 10, n. 10-10, p. 95, 2010.
- ZHONG, Z.; RYCHKOV, V.; LASTOVETSKY, A. Data partitioning on multicore and multi-gpu platforms using functional performance models. *IEEE Transactions on Computers*, IEEE, v. 64, n. 9, p. 2506–2518, 2015.