International Joint Conference on Industrial Engineering and Operations Management- ABEPRO-ADINGOR-IISE-AIM-ASEM (IJCIEOM 2020)

Processing time reduction of actuarial calculus of the Armed Forces: an application of Parallel Computing

Santos M[1], Gomes C F S[2], Martins E R[3], Costa I P A[4], Santos R C E[5]

Abstract The Armed Forces (FFAA) have a considerable mass of financial and biometric data of its pensioners that needs to be submitted to its own actuarial mathematical model, in order to obtain the result of the actuarial projection of 75 years, which is a legal obligation that must be met annually. The aim of the study is to reduce the computational times necessary to obtain the results of this projection, through the application of Parallel Computing and Software Engineering techniques. It is known that actuarial calculus is a mathematical method that uses financial, economic and probabilistic concepts to size the amount of resources and contributions necessary to pay future benefits of policyholders. The research presented here achieved significant results, in that it reduced by more than 90% the processing times of the calculations referring to the actuarial projections of FFAA pensioners. In the specific case of the Brazilian Army, from a mass of data of 500,000 records, it was possible to execute a complex mathematical model of high recursivity in about 5 minutes. Currently, the world goes through numerous transformations, generating new demands for society. Problems grow in quantity and complexity, in an increasingly dynamic and volatile context, requiring quick decisions from managers, especially at the strategic level. Hence the importance of the speed of actuarial calculus under study, because, thus, it can be timely trace dearly in order to support the decision of the FFAA Senior Management.

Keywords: Parallel Computing; Time Reduction; Actuarial Calculus; Armed Forces; Actuarial Projection.

1 Introduction

The Actuarial Calculus is a mathematical method that uses financial, economic and probability concepts to measure the amount of resources and contributions necessary to pay future benefits of the insureds of the Social Security Funds and Institutes, also called Social Security Own Regimes, which seeks the balance between the financial results and the actuarial projection. The maintenance of this balance is of paramount importance, especially in times of economic crisis, that's been causing successive budget restrictions for Brazil (Tenório *et al.*, 2020).

[1] Marcos dos Santos (✉e-mail: marcosdossantos_doutorado_uff@yahoo.com.br) PhD in Production Engineering by *Universidade Federal Fluminense* (UFF). Project Manager of the *Centro de Análises de Sistemas Navais* (CASNAV) and Professor at the *Instituto Militar de Engenharia* (IME) – Rio de Janeiro, RJ – Brazil.

[2] Carlos Francisco Simões Gomes (✉e-mail: cfsg1@bol.com.br) PhD in Production Engineering by *Universidade Federal do Rio de Janeiro* (UFRJ). Associate Professor in the Department of Production Engineering at *Universidade Federal Fluminense* (UFF) and Professor of the Graduate Program in Production Engineering at *Universidade Federal Fluminense* (TPP/UFF) – Niterói, RJ – Brazil.

[3] Ernesto Rademaker Martins (✉e-mail: radmart@yahoo.com.br) Master in Production Engineering by *Universidade Federal do Rio de Janeiro* (UFRJ) – Rio de Janeiro, RJ – Brazil.

[4] Igor Pinheiro de Araújo Costa (✉e-mail: costa_igor@id.uff.br) Master's student in Production Engineering by *Universidade Federal Fluminense* (UFF) – Niterói, RJ – Brazil.

[5] Ronaldo Cesar Evangelista dos Santos (✉e-mail: ronaldo.evangelista@marinha.mil.br) Systems Analyst and Mathematician in the Operational Research Division of the *Centro de Análises de Sistemas Navais* (CASNAV) – Rio de Janeiro, RJ – Brazil.

It is not a new discipline, which has emerged in recent centuries, but an unfolding of procedures performed since antiquity, which naturally took on a new feature with the works of Fermat and Pascal in France, De Witt in the Netherlands, Grauns and Halley in England, which advanced in the studies of probability and demographics related to human longevity (Dias and Santos, 2010).

The actuarial projections of income and expenses for a pension and pension entity are intended to quantify the estimated future costs of payment of benefits and the estimated future income from participant contributions. The projections are important for entities and governments to provision such monetary amounts for subsequent years, ensuring actuarial balance and reducing the risk of illiquidity. According to Conde (2005), when contributions are properly defined, planning tends to have a financial-actuarial equilibrium. In these cases, mathematical and/or logical models are necessary to support the decision-making process (Costa *et al.*, 2020; Santos *et al.*, 2015).

Pension and social security institutions project future costs and revenues over a 75-year time horizon, however, nothing prevents the projection from being calculated at 80 or 100 years. The main assumptions that impact on this actuarial calculation are: mortality table of invalids, table of mortality of invalids, table of disability, turnover rate, wage growth, rate of inflation and replacement of assets.

In Brazil, it is mandatory to carry out the annual actuarial projection of 75 years by state companies and/or their pension funds. Therefore, investments are needed in the development of technologies necessary to obtain results or the contracting of third-party services. In both situations, however, the use of traditional programming techniques, that is, without the use of parallel computing, even if the best practices of software engineering are used, leads to the production of results that, although of good accuracy, can still be improved in terms of computational performance.

2 Background

In the development of the remuneration studies, the Ministry of Defense (MD) realized that the lack of detailed information on military payroll expenses was the major hindrance to negotiations with the Ministry of Planning, Budget and Management (MPBM) and, consequently, the major problem for the monitoring and evaluation of the financial impact of the proposed adjustments or of changes in the legislation in force. In order to reduce this deficiency and make it possible to monitor the effects of Provisional Measure (PM) nº 2.215 / 2001 (Brasil, 2001), SMIB was created.

Faced with the limitations of sequential computing, we opted for Parallel Computing, so that the results of the actuarial projections of the Armed Forces could be generated in an acceptable time, in order to support the Senior Administration of the Armed Forces in matters related to Pension Reform with the Government Federal.

In order to support the development and use of a computational application for the actuarial calculation, we sought knowledge in areas related to Parallel Computing, focusing on the creation of a computational environment capable of propitiating the actuarial projections in a timely manner. Parallel Computing is described by (Gottlieb and Almasi, 1989) as a kind of computing in which multiple calculations are performed in several parallel, rather than sequential, processing units. The idea here is to replace a large mass of processing in a single CPU by smaller processing in several CPUs, as shown in Fig. 1.

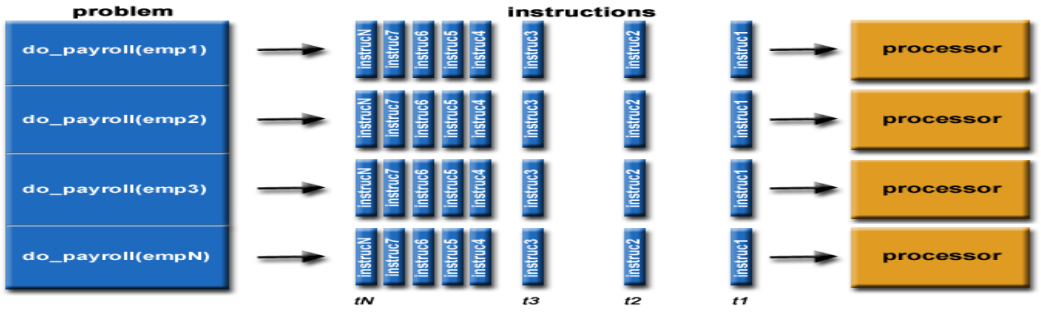

Fig. 1. Processing a problem through Parallel Computing. Source: Adapted from (Barney, 2010).

The adoption of a methodology that uses parallel computing resources in the development of an application to calculate actuarial projections can, by extrapolating the results achieved within the scope of the Armed Forces, considerably shorten the time needed to obtain results and reduce the costs of organizations with the contracting of third-party services.

2.1 Parallel Programming with C#

Current architectures may contain two, four, six or more complex processing cores, and are very much employed in exploring parallelism at the instruction and thread level. In theory, such architectures could be extended to tens or even hundreds of cores in the future, but they run into two main obstacles: the difference between processor and memory speeds, known as memory walls, and excessive heat generation due to high frequencies necessary for the operation of such architectures (Hwang *et al.*, 2013).

Multi-core processors (color) have been on the market for many years and are currently available on most devices. However, many developers continue to do what they have always done: create programs that use a single stream of isolated sequential control inside a program called a thread. This means that not all processing power available on the machine is used. Most currently commercialized computers have a four-core and four-core processor, or six-core and eight-core processors. By purchasing a computer with such features, users pay for extra processing power. However, by providing a program that uses a single thread, the developer does not allow the use of this extra processing capacity.

According to (Kirk, 2010) and (Golov and Rönnbäck, 2017), it is a simple task to achieve a ten-fold speed up when an application makes use of data parallelism. A computation is said to be parallel when a program runs on a multiprocessor machine in which all processors share access to available memory, i.e. the same address on different processors corresponds to the same memory location.

Multi-threaded processing is not new to experienced C # developers, but it is not always easy to develop programs that use all the available processing power. In addition, the evolution of programming languages has facilitated the work of developers by simplifying the implementation of parallel programming. In order to advance the use of parallel programming, it is important to establish two important concepts: synchronous execution and asynchronous execution. A good insight into these two modes of execution is basic knowledge to improve the performance of your applications. When executing a synchronous operation, the program executes all the tasks in sequence. When firing the execution, each task will only be executed after the previous task ends.

When executing an operation in asynchronous mode, the program triggers the tasks, when necessary, and they are started and closed concomitantly to the execution of other tasks. As it induces image analysis, the same tasks running asynchronously will require a shorter execution time than when executed in a synchronous manner, for the simple fact that there is no need for one task to "wait" for another task to start. Waiting, in this case, would only be acceptable if there were a dependency relationship between the tasks, that is, if a task depended, for example, on a calculation that another task, still in execution, is producing.

From the previous paragraph it is possible to deduce that the use or not of the parallel programming should be evaluated by the developer. In some situations, its use can be quite beneficial by decreasing the execution times and producing the results. In other situations, however, its unbalanced use may even degrade application performance. Since the implementation of parallel programming with C # requires the explicit insertion in the code of specific instructions of parallels, it is up to the developer to decide when and where to use such instructions.

The question may arise as to why many developers still choose to run synchronously if asynchronous execution takes less time. The answer to such a question is not simple. What can be said straight away is that with asynchronous programming, the developer has some new challenges:

• Synchronize tasks. Assuming that it is necessary to start a task only after the other two are finished. It can be necessary to create a wait mechanism to wait for all tasks to finish before performing the new task;

• Solve competition problems. If a shared resource exists, such as a list that is written to a task and read into another task, you would need to create a mechanism to ensure that the list is kept in a known state;

• Adapt to a new programming logic, since there is no logical sequence. Tasks can end at any time and you no longer have control of which ends first.

Asynchronous programming requires a paradigm shift on the part of the developer. Its adoption, however, has some advantages. One of the most significant is the non-crash of the user interface (UI), as the tasks can be executed in the background. Another advantage is the ability to use all the cores of the machine, making better use of its resources.

2.2 Asynchro Programming with Async and Await

It is possible to avoid performance bottlenecks and improve the overall response of software using asynchronous programming. However, traditional techniques for writing asynchronous applications can be tricky, making it difficult to develop, debug, and maintain.

Asynchrony is essential for activities exposed to a potential block, such as when the application accesses the web. Access to a web resource is sometimes slow or subject to delays. If such activity is blocked within a synchronous process, the entire application will be on hold. In an asynchronous process, the application can continue to perform another task, which does not rely on the web resource, until the task exposed to a potential block ends.

Async and await, keywords in C #, are the core of asynchronous programming. Using these two words, one can use the features of the .NET Framework or the Windows runtime to create an asynchronous method almost as easily as creating a synchronous method.

Fig. 2 shows part of an asynchronous routine. When using the async keyword, you can write your code the same way you write the synchronous code because the compiler takes care of all the complexity and frees the programmer to write program logic.

```
async Task<int> AccessTheWebAsync()
{
    HttpClient client = new HttpClient();
    Task<string> getStringTask = client.GetStringAsync("http://msdn.microsoft.com");
    DoIndependentWork();
    string urlContents = await getStringTask;
    return urlContents.Length;
}
```

Fig. 2. Asynchronous Routine.

To use this routine, you should expect your return using the await keyword, as in the example: "string urlContents = await client.GetStringAsync ()". Following these guidelines, when the compiler encounters a method with the expression await, which marks the point at which the method cannot continue until the expected asynchronous operation completes, it starts running in the background and continues to run other tasks. When the routine is complete, execution returns on the instruction following the routine call.

C # language enhancements with the async and await keywords restore the sequential order of code while using system resources efficiently. There are still some relevant aspects to be observed, such as concurrency or task synchronization, but these are minor, compared to the work required to create a good program that uses parallel processing. Parallelism is to divide in order to multiple processors that work concurrently; parallelism computing tasks and accelerates the efficiency (Laili *et al.*, 2019). The use of the techniques described here helps a lot in the creation of programs that make use of parallel computing and that, therefore, better use the resources of the system.

2.3 Actuarial Premises

In recent decades there has been significant growth in the amount of information stored in electronic formats. According (Szalay *et al.*, 2000), the amount of information in the world doubles every 20 months. This was provided by several factors, such as falling prices for storage and processing equipment, and advances in data capture and generation mechanisms, such as barcode readers, remote sensors, and space satellites.

According to Piatetsky-Shapiro (1990), volumes of data, produced and stored on a large scale, are unfeasible to be read or analyzed by experts, using traditional methods such as spreadsheets and operational reports. On the other hand, it is known that large amounts of data equate to a greater information potential.

The spreadsheets constitute tools used on a large scale by actuarial sciences professionals. During the work of defining actuarial and financial premises, sometimes the actuary has the need to manipulate large volumes of data to produce basic knowledge about the population studied.

Actuarial and financial premises represent a formal set of estimates for events: biometric, financial, economic, demographic and social. The choice and use of actuarial premises uncompromised with the reality to which participants, sponsors and entities are submitted may lead to incorrect costs, causing deficit or technical overexposure, as well as underexposure to risks.

The use of more conservative assumptions can drive the initial costs higher, albeit with lower risks of rising costs. However, the adoption of less conservative assumptions should be made with the knowledge of the risk that they may not be confirmed, allowing for critical solvency problems in the future (Rodrigues, 2008).

The actuarial premises will always be criteria permeated by common sense, because excesses of safety margins are as burdensome as the excess risks that are intended to be assumed. Both lead to the inability to pay, either by a participant and/or sponsors, either from the pension entity itself or benefit plan. Due to the above, the indication emerges that spreadsheets may not, in some cases, be the ideal tool to be used by an actuary to define actuarial and financial premises.

3 Implementation of Parallel Loading of Databases

In the actuarial study of the Armed Forces, the actuarial assumptions are obtained from a population superior to a million and a half of registries. In a scenario with a data volume of this magnitude, a Database Management System (DBMS) is an ideal tool for manipulating and storing the data involved, not only in the definition of the actuarial assumptions, but also for the realization of the actuarial calculation itself. Hence the study presented here has a considerable applicability, since it can be used in the actuarial calculation of any other institution.

A Database Management System (DBMS) is a set of programs that manage the structure of the database and control access to the stored data (Coronel and Peter, 2010). To some extent, the database resembles an electronic file with very well-organized content with the help of robust software, known as DBMS.

Although formally defined the DBMS does not have, in the scope of this work, the role of principal actor. Thus, regardless of the technical characteristics of the DBMS used, the use of C # and Parallel Computing techniques are enough to greatly reduce the computational effort involved in the actuarial calculation of the Brazilian Armed Forces.

Based on the concept that, in many cases, the actuarial assumptions are raised by observing a large mass of data and that a DBMS is shown as an ideal tool for the manipulation and storage of the same, it presents a practical application the use of parallel computing to insert records into a database. The first step is to create the table in the database where the data will be inserted. It should be noted that SQL syntax is compatible with Microsoft SQL Server 2008, but can be easily adapted to the standard used by other databases.

The Task Parallel Library (TPL) library supports data parallelism through the System.Threading.Tasks.Parallel class. This class provides parallel implementations based on for loop and foreach methods. Write the logic of a for or foreach parallel loop in the same way that you write a sequential loop. You do not have to create threads or work queues. In basic loops, there is no need to worry about locks. TPL handles all low level work.

When a parallel loop is executed, TPL partitions the data source so that the parts are operated in simultaneous loops. Internally, the task scheduler partitions the task based on the workload and system resources. When possible, the scheduler redistributes work between multiple threads and processors if the workload becomes unbalanced. In the construction of this example, a small database containing 81,000 (eighty-one thousand) records was used, with each record containing 15 (fifteen) fields. For purposes,

initially, merely illustrative, the application was endowed with an extremely simple interface, containing only two buttons: Sequential Test and Parallel Test.

Through the encodings coupled to each of these buttons, it is possible to verify and compare the computational effort when using the sequential approach and when using the parallel approach. Fig. 3 shows the encoding of the Sequential Test button, while Fig. 4 shows the encoding of the Parallel Test button.

```
private void button1_Click(object sender, EventArgs e)
{
    t_inicio = DateTime.Now;
    TMilitarAtivoDAO vMilitarAtivoDAO = new TMilitarAtivoDAO();
    TListaMilitaresATIVOS vLista = new TListaMilitaresATIVOS();
    vLista = vMilitarAtivoDAO.ListarMilitaresAtivos();
    // Versão sequencial
    foreach (TMilitarATIVO vMilitarATIVO in vLista)
    {
        InserirMilitar(vMilitarATIVO);
    }
    t_fim = DateTime.Now;
    t_diferenca = t_fim.Subtract(t_inicio);
    MessageBox.Show(t_diferenca.TotalSeconds.ToString("0.000000") + " segundos");
}
```

Fig. 3. Coding of the sequential test button.

```
private void button2_Click(object sender, EventArgs e)
{
    t_inicio = DateTime.Now;
    TMilitarAtivoDAO vMilitarAtivoDAO = new TMilitarAtivoDAO();
    TListaMilitaresATIVOS vLista = new TListaMilitaresATIVOS();
    vLista = vMilitarAtivoDAO.ListarMilitaresAtivos();
    // Versão Paralela
    Parallel.ForEach(vLista, vMilitarATIVO => InserirMilitar(vMilitarATIVO));
    t_fim = DateTime.Now;
    t_diferenca = t_fim.Subtract(t_inicio);
    MessageBox.Show(t_diferenca.TotalSeconds.ToString("0.000000") + " segundos");
}
```

Fig. 4. Coding of the parallel test button.

Clicking the Sequential Test and Parallel Test buttons (at different times) produces the results shown in Fig. 5, respectively. The simple observation of the results shows that the change made in programming with the use of parallel programming was enough to decrease from 34.27s to 14.80s the time required to insert the 81,000 records of the example in question in the database, no matter what this database is.

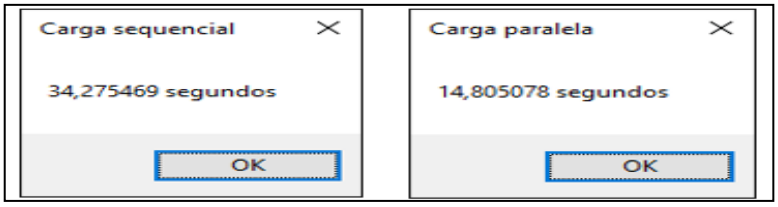

Fig. 5. Processing time on each load.

The practical application was developed and executed in a relatively simple machine with only 8 GB of RAM and an Intel® Core ™ i3-4150 Processor (3M Cache, 3.50 GHz), with 2 cores and 4 threads.

4 Results

The work developed focused on optimizing the computational application of the actuarial calculation for the production of the actuarial projections of the pensioners of the Armed Forces. In this sense, any and all access to the hard disk (HD) was eliminated during the execution of the calculations.

In order to achieve this, a technique has been developed which loads all the data necessary for the calculations into the computer memory as soon as the application is started. During the application load the data is read from the respective database tables and is generated from SQL queries, and persisted in structures (arrays) in memory. Once loaded, the data is only read in these structures until the application is closed, which makes the data reading and writing processes very fast.

This technique proved to be efficient, due to the fact that the HD is much slower than RAM. While a DDR4-2400MHZ (1606R) module communicates with the processor at a theoretical speed of 4200 MB / s,

the sequential read speed of the current HDs hardly exceeds the 233 MB / s mark. In addition to this, the HD access time, ie the time required to locate the information and initiate the transfer, is considerably higher than that of the RAM.

While memory is spoken at access times of less than ten nanoseconds, most HDs work with access times greater than ten milliseconds. This causes the HD performance to be much lower when reading small files scattered around the disk (as is the case with virtual memory). In many situations, the HD gets to the point of not being able to handle more than two or three hundred requests per second. From the foregoing, it can be understood that once the loaded memory arrays already allow their data to be accessed at high speeds, compared to the speeds of access to the hard disk. And when the data access methods of such structures are developed using parallel programming techniques, the time required to produce the results is considerably reduced.

In the example in Fig. 6, vList is a structure that contains all active military loaded from the database when the application was started, being used as data source and traversed by the parallel loop. An implementation is therefore available that accesses data structures, in this case a data repository, in memory using parallel programming techniques.

```
// Abordagem Paralela
Parallel.ForEach(vLista, vMilitarATIVO => InserirMilitar(vMilitarATIVO));
```

Fig. 6. Processing time on each load.

Table 1 shows the time required for the production of results by the application of the actuarial calculation in two versions. The first version, "Previous Version" does not contemplate any of the techniques, patterns or features presented in this chapter. The second version, called "Refactored Version", is the final product of the work presented here. In addition, it shows that in all groups analyzed, there was a significant reduction in the time needed to produce the results of the actuarial calculation, notably in the Actuarial Calculation of Present Value.

The data are presented in three distinct groups, these groups being notably those that require longer processing time, and whose results are of interest to the study under study.

Table 1. Comparison of processing times.

Process	Previous version [Approximate] HH:MM:SS	Refactored Version [Approximate] HH:MM:SS	Time Reduction (%)
Importing databases.	04:00:00	00:08:00	96.7%
Actuarial calculation of Present value.	23:00:00	00:07:00	99.5%
Actuarial projection with a term of 75 years.	16:00:00	00:05:00	99.5%
TOTAL TIME	**43:00:00**	**00:20:00**	**99.2%**

In this context, the use of Parallel Computing techniques and resources proved to be a solution to reduce the computational effort, allowing the use of all the available processing capacity in the computers.

5 Final Considerations

The world goes through many transformations, generating new demands for society. The problems grow in quantity and complexity, in an increasingly dynamic and volatile context, requiring rapid decisions by managers, especially at a strategic level. Hence the importance of the quickness of the actuarial calculation of pensioners of the Armed Forces, since, in this way, it is possible to quickly trace numerous scenarios in

order to support the decision of the High Administration of the Armed Forces. It should be noted that such decisions can directly affect the lives of more than 2,000,000 Brazilian citizens.

Thus, in order to accelerate the calculation of actuarial projections, Parallel Computing was used, reducing by more than 90% the processing times of the calculations referring to the actuarial projections of the pensioners of the Armed Forces. In the specific case of the Brazilian Army, from a data mass of 500,000 records, it was possible to execute a complex mathematical model of high recursion in about five minutes. It should be emphasized that the methodology proposed by this research can be applied in other actuarial problems.

References

Barney, B. (2010), "Introduction to parallel computing", *Lawrence Livermore National Laboratory*, Vol. 6 No. 13, p. 10.

Brasil. (2001), "Dispõe sobre a reestruturação da remuneração dos militares das Forças Armadas, altera as Leis nos3.765, de 4 de maio de 1960, e 6.880, de 9 de dezembro de 1980, e dá outras providências", *Medida Provisória No 2.215-10, de 31 de Agosto de 2001*, available at: http://www.planalto.gov.br/ccivil_03/mpv/2215-10.htm (accessed 7 October 2020).

Conde, N.C. (2005), "Os parâmetros atuariais dos planos de previdência complementar", *Revista Fundos de Pensão*, Vol. 24 No. 309, pp. 79–83.

Coronel, C. and Peter, R. (2010), "Sistemas de banco de dados: projeto, implementação e administração", *São Paulo: Cengage Learning*.

Costa, I.P. de A., Maêda, S.M. do N., Teixeira, L.F.H. de S. de B., Gomes, C.F.S. and Santos, M. dos. (2020), "Choosing a hospital assistance ship to fight the Covid-19 pandemic", *Revista de Saude Publica*, Universidade de Sao Paulo, Vol. 54, available at:https://doi.org/10.11606/S1518-8787.2020054002792.

Dias, C.R.B. and Santos, J. dos. (2010), "Gestão atuarial dos regimes próprios de previdência social", *Apostila de Curso de Pós-Graduação Em RPPS. Recife: Centro Brasileiro de Estudos Previdenciários*.

Golov, N. and Rönnbäck, L. (2017), "Big Data normalization for massively parallel processing databases", *Computer Standards & Interfaces*, Elsevier, Vol. 54, pp. 86–93.

Gottlieb, A. and Almasi, G. (1989), *Highly Parallel Computing*, Benjamin/Cummings Redwood City, CA.

Hwang, K., Dongarra, J. and Fox, G.C. (2013), *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, Morgan Kaufmann.

Kirk, D.B. (2010), "W. mei W", *Hwu, Programming Massively Parallel Processors. Morgan Kauffman*.

Laili, Y., Zhang, L. and Li, Y. (2019), "Parallel transfer evolution algorithm", *Applied Soft Computing*, Elsevier, Vol. 75, pp. 686–701.

Piatetsky-Shapiro, G. (1990), "Knowledge discovery in real databases: A report on the IJCAI-89 Workshop", *AI Magazine*, Vol. 11 No. 4, p. 68.

Rodrigues, J.A. (2008), *Gestão de Risco Atuarial*, Saraiva São Paulo.

Santos, M. dos, Quintal, R.S., da Paixão, A.C. and Gomes, C.F.S. (2015), "Simulation of operation of an integrated information for emergency pre-hospital care in Rio de Janeiro municipality", *Procedia Computer Science*, Elsevier, Vol. 55, pp. 931–938. Available at:https://doi.org/10.1016/j.procs.2015.07.111.

Szalay, A.S., Kunszt, P.Z., Thakar, A., Gray, J., Slutz, D. and Brunner, R.J. (2000), "Designing and mining multi-terabyte astronomy archives: the Sloan Digital Sky Survey", *ACM SIGMOD Record*, ACM New York, NY, USA, Vol. 29 No. 2, pp. 451–462.

Tenório, F.M., dos Santos, M., Gomes, C.F.S. and de Carvalho Araujo, J. (2020), "Navy Warship Selection and Multicriteria Analysis: The THOR Method Supporting Decision Making", *International Joint Conference on Industrial Engineering and Operations Management*, Springer, pp. 27–39. Available at:https://doi.org/10.1007/978-3-030-56920-4_3.