STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **10**, February 2022, pp 192–203.
Published online in International Academic Press (www.IAPress.org)

MCDM Filter with Pareto Parallel Implementation in Shared Memory Environment

Loubna Lamrini [1,*], Mohammed Chaouki Abounaima [1], Fatima Zahra El Mazouri [2], Mohammed Ouzarf [1], Mohammed TALIBI ALAOUI [1]

[1]*Laboratory of Intelligent Systems and Applications, Faculty of Sciences and Technologies, Sidi Mohammed Ben Abdellah University, Fez, Morocco*
[2]*Université Polytechnique Hauts-de-France, LAMIH UMR CNRS 8201, Valenciennes, France*

Abstract Nowadays, multi-criteria decision-making (MCDM) methods are often used to solve problems involving large data sets, especially with the advent of the big data age. In such a context, the multi-criteria decision-making methods theoretically can be used but technically are not efficient in terms of the treatment time. Indeed, the majority of commercial or even experimental multi-criteria decision support tools always have limits in terms of the number of alternatives and the number of criteria to be retained in the decision-making process, which presents a computational challenge to relieve. This present paper discusses the application of parallel computation to meet this challenge and make the application of MCDM methods possible in the presence of a big number of alternatives and criteria. More precisely, the main objective of this work is to provide a parallel filtering mechanism that can be executed even on accessible personal computers and offering a short and reasonable response time. The introduction of a filter as a first step in the decision-making process consists in retaining, as alternatives to be treated by the MCDM method, and by parallel processing only the Pareto solutions. To achieve this objective, we propose a parallel computing approach deploying the Open MP (Open Multi-Processing) paradigm on a shared memory environment to find Pareto solutions. To prove the effectiveness of the proposed approach for problems with large dimensionality, several numerical examples with different dimensions will be examined.

Keywords Multi-Criteria Decision Making/Aiding (MCDM/MCDA), Decision process,Pareto solution, Pareto Front, Open Mp paradigm, shared memory environment.

AMS 2010 subject classifications 94A08, 68U10

DOI:10.19139/soic-2310-5070-1216

1. Introduction

This present article is an extension of our paper published at the international conference [1]. In this paper, we have shown the importance of reducing the dimension of the decision matrix used by any MCDM method. We have proposed a preliminary filter based on the Pareto front to eliminate low potential alternatives from the beginning of the decision process. We have proposed two algorithms to reduce the number of pairwise comparisons. Knowing that in the context of big data, the execution time of pairwise comparisons is very long [2]. In this paper, we propose a parallel implementation to plot the Pareto front using thread based parallelism on a multi-core shared memory computer.

The big data era's evolution goes into different shapes such as volume, velocity, variety, veracity; all are challenging tasks. Decision-making based on multi-criteria is one of the most critical issues solving ways to select the most suitable decision among a large volume of data.

[*]Correspondence to: Loubna Lamrini (Email: loubna.lamrini@usmba.ac.ma). Faculty of Sciences and Technologies, Sidi Mohammed Ben Abdellah University, Road of Imouzzer, 2202, Fez, Morocco.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

Many software has been developed for solving MCDM problems; such DECERNS [3], Triptych, SANNA. For example, the Triptych package for TOPSIS analysis is developed and distributed by Statistical Design Institute, with a proprietary commercial license, limits the number of criteria and actions to 200. SANNA limits the number of criteria to 50 and alternatives to 20. TechSelect is a scenario-based decision support tool with limitations in terms of the attributes and alternatives in the decision matrix [4]. Our goal is to provide a filter that reduces a large set of alternatives to a smaller set most likely contains the best choices.

The general aim of decision making in the era of Big data is to reduce large-scale problems to a scale that humans can comprehend and act upon. Decision-makers cannot count anymore on classical MCDM methods to analyze data-sets with the profusion of alternatives available.

Parallel computing is very promising to speed up the computation and meet the growing demand for resources to treat and deal with a massive amount of data. We can also refer to a few applications of parallel processing to multi criteria decision making [5]. We attempt to exploit all the available computing power usefully. Especially that all processors are now multi-core, and it is essential to take advantage of this fact. Parallel computing is leveraged through the increasing availability of powerful computing capabilities.

The rest of this paper is organized as follows: the next section will introduce the background of multi-criteria decision making. The third section presents the parallel computing technologies trend. Section 4 illustrates the proposed approach. Finally, the last section will summarize the result and computational performance reached. We conclude this paper by providing a summary and conclusions as well as suggests future research topics.

2. Background

MCDA reduces human error instead of relying on intuition and experience for effective decision-making. In this section, the fundamental of MCDM is revisited as well as some problems, and the parallel computing technology Open MP is introduced.

2.1. Multiple criteria decision analysis

An MCDA or MCDM problem consists of judging a set of alternatives based on their evaluations on a set of criteria. In MCDA, three decision problems are possible [6].

- The first allows to decision-maker (DM) to ranking the alternatives from the best to worst alternative,
- The second consists of sorting the alternatives into predefined categories,
- The third problem allows DM to select and choose the best alternative.

For the resolution of the third MCDA problems, several methods have been proposed in the literature; which we can classify into two main categories of approach [6] [7]: the first concerns the approach of the synthetic criterion, where all the criteria are aggregated into a single criterion, as examples of these methods, we cite the Weight Sum Method, the Weight Product Method, the Goal Programming method, the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS), etc. The second approach is known as the outranking approach, where we build a binary relationship between the alternatives, called the outranking relationship. As examples of the methods of this approach, we cite the two popular methods ELECTRE (ELimination et Choix Traduisant la REALITE) [7] and PROMETHEE (Preference Ranking Organization METHod for Enrichment of Evaluations) [8].

Any MCDA problem considers a set A of n alternatives, consisting of candidate solutions, and a family F of m criteria, the view's points according to which the alternatives will be examined and compared. To take into account, the differences between the relative importance of criteria in the decision making, a set W of weights is also provided. To summarize all the data of a decision problem, a matrix, see table 1, called performance matrix, or decision matrix, is constructed.

Table 1. Decision Matrix

Decision Matrix						
Criteria		Criteria 1	...	Criteria j	...	Criteria m
Weights		g1	...	gj	...	gm
		W1	...	Wj	...	Wm
		Min/Max	...	Min/Max	...	Min/Max
Alternatives	a1	g1(a1)	...	gj(a1)	...	gm(a1)
	...	...	...	...	...	...
	ai	g1(ai)	...	gj(ai)	...	gm(ai)
	...	...	...	...	...	...
	an	g1(an)	...	gj(an)	...	gm(an)

- $F = (g_1, g_2, ..., g_j, ..., g_m)$ is the family of m criteria $m \geqslant 2$
- $A = (a_1, a_2, \ldots, a_j, ..., a_n)$ is the set of alternatives.
- $W = (w_1, w_2, \ldots, w_j, ..., w_m)$ is the weight vector reflecting the relative importance of the criteria.
- $g_j(a_i)$ is the evaluation of the criterion g_j for the alternative a_i.
- Min indicates that the criterion to be minimized
- Max indicates that the criterion to be maximized

2.2. *Problem and restriction*

In the outranking approach, the construction of relations between the set of alternatives is an operation whose complexity depends both on the set size (number of alternatives to be considered) and the criteria to be taken into account. The main restriction of this approach resides in a large number of pairwise comparisons between the alternatives to be made to construct the outranking relationship. This fact makes the decision process, especially in massive data, very greedy for computation and memory time.

As far as we know, few references can be found about parallel implementations of Multicriteria Decision Support Method [9], even less with the Open MP paradigm. Some works focus on very specific methods : Electre III [10], AHP [11], PROMETHEE [12]. They proposed a parallel implementation that requires a very complex and expensive IT hardware infrastructure.

In [1] we have proposed a revised MCDA approach where we suggest filtering all candidate alternatives and keeping only the most relevant. For that, we propose to reject all the dominated alternatives, according to the applied dominance relation, and keep only the alternatives that are not dominated by any other alternative. The filter consists of finding the actions belonging to the Pareto front to be retained for the rest of the decision process. This operation requires several pairwise comparisons that are very computationally intensive for a large set of alternatives.

Parallel computing on modern computers shows excellent promise for speeding up computing. The key problem is how to convert the sequential algorithms proposed in [1] to parallel algorithms that properly exploit the available resources.

2.3. *The Pareto-optimal solution*

In the literature [13], there are two definitions to express the optimal situation in the sense of Pareto. Let be a set of alternatives A of size n, and considering F the set of m criteria, and let P be the set of Pareto-optimal solutions, called the Pareto front. Then an alternative a of A is a solution of Pareto Optimal and which will be considered in the Pareto Front P if and only if:

- P1 : Weak definition

$$\{b \in A \setminus g_j(b) > g_j(a) \forall g_j \in F)\} = \varnothing \tag{1}$$

- P2 : Strong definition

$$\{b \in A \setminus g_j(b) \geq g_j(a) \forall g_j \in Fand \exists g_k : g_k(b) > g_k(a))\} = \varnothing \tag{2}$$

In this paper, we use the strong definition P2 to define the Pareto-optimal solutions. The weak definition is too rigid, and it can give an empty set of elements.

The dominance relationship

The dominance relationship is noted by $a\Delta b$, and is read: a dominates b and b is dominated by a.

$$a\Delta b \Leftrightarrow \{g_j(a) \geq g_j(b) \forall g_j \in Fand\ \exists g_k : g_k(a) > g_k(b)\} \tag{3}$$

Example of the Pareto Front

To illustrate the definitions of Pareto-optimal solutions, non-Pareto-optimal solutions, and the Pareto Front, we give a case presented in figure 1. It is an MCDA problem where two criteria g1 and g2, are to be maximized. According to this example, all the choices located on the red colored curve are Pareto-optimal solutions, in the sense that there is no other choice, which is their best on the two criteria, g1 and g2. For example, choices a and b are two Pareto-optimal solutions because any other alternative does not dominate a and b. However, the choice c is not a Pareto-optimal solution because it is at least dominated by a and b: $a\Delta c$ and $b\Delta c$. As this example illustrates, the choices in gray color are not Pareto-optimal solutions because there are choices from the Pareto Front, which dominates them. As proposed and recommended by this work, the investigation of the best choices will restrict to the Pareto Front, which means that all non-Pareto-optimal options will be excluded from the next step of the decision process.

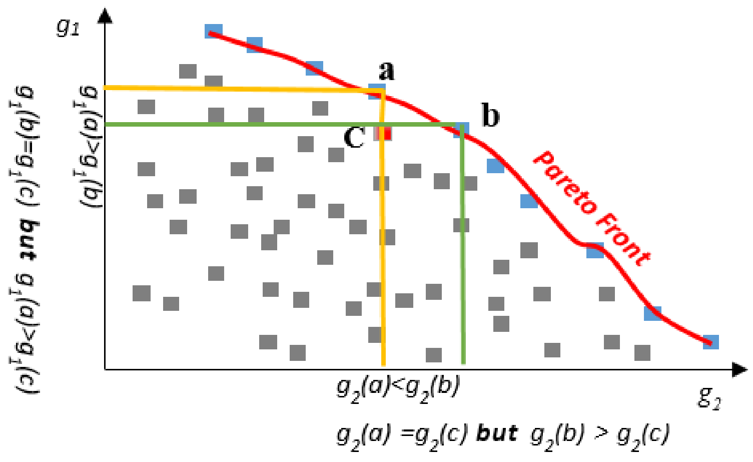


Figure 1. Example of the Pareto Front.

3. Parallel computing and the evolution of multicores

The use of parallel processing is today essential for solving practical problems in science and engineering. Parallelism is a way of speeding up computations that make high time and require large memory, especially for massive data.
In the past, for a very long period, programmers do not take care of the execution time. They rely on evolving

hardware and improving CPU speed. Moore's Law [14] states that processing power doubles every 18 months. This law remained correct throughout the 90s decade. This was the result of improvements in the gates-per-die count of transistors per area, the main attribute of CPUs that Moore based his law on. The number of instructions executed per time unit (clock speed) and the instruction-level parallelism (ILP), basically meaning the possibility of performing more than just one single operation within the same clock. Unfortunately, Moore's law of performance gain is over [15]. In recent years CPU manufacturers have started selling CPUs with more computational cores instead of faster CPUs, as beginning in 2003, the laws of physics put an end to the increase in clock speed. The principal reason for this is that doubling the clock speed solicits halving the distance travelled by the electrical signal per clock cycle, which requires the physical size of the CPU to be twice as small [16]. However, reducing the physical dimensions of CPUs is limited by the diffraction limits of the lithographic methods used for chip manufacturing [17]. Other methods are used to increase performance that can at least partly compensate for the limited increase in clock speed. These are, for example, sophisticated ILP schemes, which nowadays are the main basis for performance improvement apart from the gate count. These methods are what manufacturers focus on today, resulting in feature-rich CPUs that are additionally equipped with an increasing number of computational cores.

For the next decade, with the multi-core revolution, Major processor vendors have shifted the trend from increasing clock speeds to adding on-chip parallelism support with multi-core processors. In other words, put more than one CPU core on a single chip [18]. This effectively makes a system with a processor with two cores operate like a dual-processor computer, and a system with a processor with four cores operate like a quad-processor computer.

Based on the computer's memory architectures classification, there are shared-memory computers, distributed-memory computers (clusters), and hybrid supercomputers. For each type of parallel architecture corresponds one or more programming paradigms, with associated languages [19]:

1. Distributed-memory computer: requires a communication network to connect inter-processor memory because each processor, local memory operates independently. The memory is scalable with the number of processors, but data communication between processors is more complicated. The paradigm MPI is a communication protocol for parallel programming in distributed-memory architecture. It is used to allow applications to run in parallel across several separate computers connected by a network.

2. Shared-memory computer: has the ability for all its processors to operate independently but share the same global memory resources. Changes in one memory location affected by one processor are visible to all other processors. Therefore, data sharing between tasks is both fast and uniform due to the proximity of memory to central processing units (CPUs). Still, the scalability between memory and CPUs is relatively poor. the Open MP approach is a development paradigm for this type of architecture.

3. Hybrid, heterogeneous systems: combinations of several parallelism models: MPI + Open MP + GPUs accelerators, etc.

Moreover, to take advantage of multi-core power, we must adapt our programs and make their execution distributed over the different cores available. This tendency motivated us to favor using a parallel implementation based on the cooperation of threads to filter the initial set of alternatives. Currently, the majority of laptops are shared-memory computers which prompted us to use the Open MP paradigm.

3.1. Open MP paradigm

The Open MP standard was formulated in 1997 as an API for writing portable, multithreaded applications [20]. It started as a Fortran-based standard but later grew to include C, C++, and others. The current version is Open MP 5.1 [21]. Open MP is among the most widely used parallel programming (API) and is largely directive-based, making it one of the most accessible parallel programming APIs.

3.2. Parallelism in Open MP

Open MP supports the fork–join model of parallel computing where at particular points in the execution, the master thread spawns a number of threads that are executed concurrently to gain a performance benefit. An Open MP program begins with a single thread, the master thread. As the program executes, the application may encounter parallel regions in which the master thread creates thread teams. At the end of a parallel region, the thread teams are parked and the master thread continues execution, see figure 2. From within a parallel region, there can be nested parallel regions where each thread of the original parallel region becomes the master of its own thread team. Nested parallelism can continue to further nest other parallel regions [20]. Fork–join model is a method of programming on parallel machines in which one or more child processes branch out from the root task when it is time to do work in parallel and end when the parallel work is done [22].

The principle is as follows: we introduce directives in a sequential program, and these help the compiler to build parallel code. Open MP is not a language in itself. It provides a set of directives (pragmas), routines and environment variables (Figure 3 illustrates the environment of Open MP).

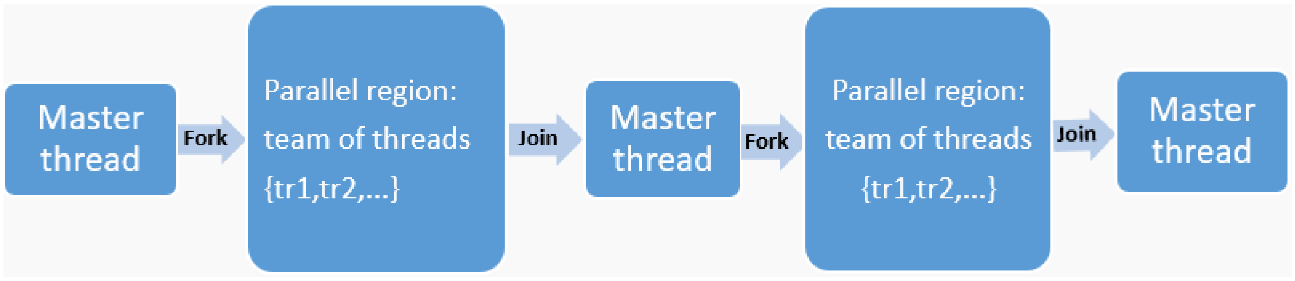

Master thread
Fork
Parallel region: team of threads {tr1,tr2,...}
Join
Master thread
Fork
Parallel region: team of threads {tr1,tr2,...}
Join
Master thread

Figure 2. Open MP Fork Join model

The environment variable OMP_NUM_THREADS defines the number of threads, but it can be modified in the program using the NUM_THREADS clause. The number of threads is not necessarily equal to the number of physical cores available.

The Open MP API for C / C ++ contains a large number of directives and constructions [20]. Among the most important directives, we cite :

#pragma omp parallel: Define a parallel region.
#pragma omp for: Tells Open MP that the for loop, when called from a parallel region, should have its iterations divided among the thread team.
#pragma omp sections: Work sharing (instruction blocks).
#pragma omp single: Execution by a single thread.

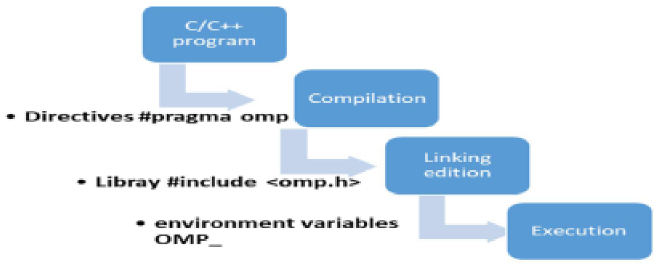


Figure 3. Open MP API

4. Proposed approach

4.1. Methodology

The primary novelty and contribution of the current research study is to introduce, into the decision process, a preliminary filtering step based on parallel computation to subsequently make the decision-making process practical and operational to study several scenarios of massive structured data problems.

We propose a decision process revision that considers the scenario of DMs with a large number of alternatives (facing large amounts of data). The contributions of this paper can be mainly summed up as two aspects:

1. Introduce a filter based on the Pareto front as a first step in the decision process. The proposed filter reduces a large set of alternatives to a smaller set that most likely contains the best choice.

2. The applicability of Open MP, the dominant shared-memory parallel programming model in high-performance computing, to filter alternatives with Pareto front. Parallel implementation significantly reduces execution time compared to the previous sequential implementation.

4.2. Solution Implementation

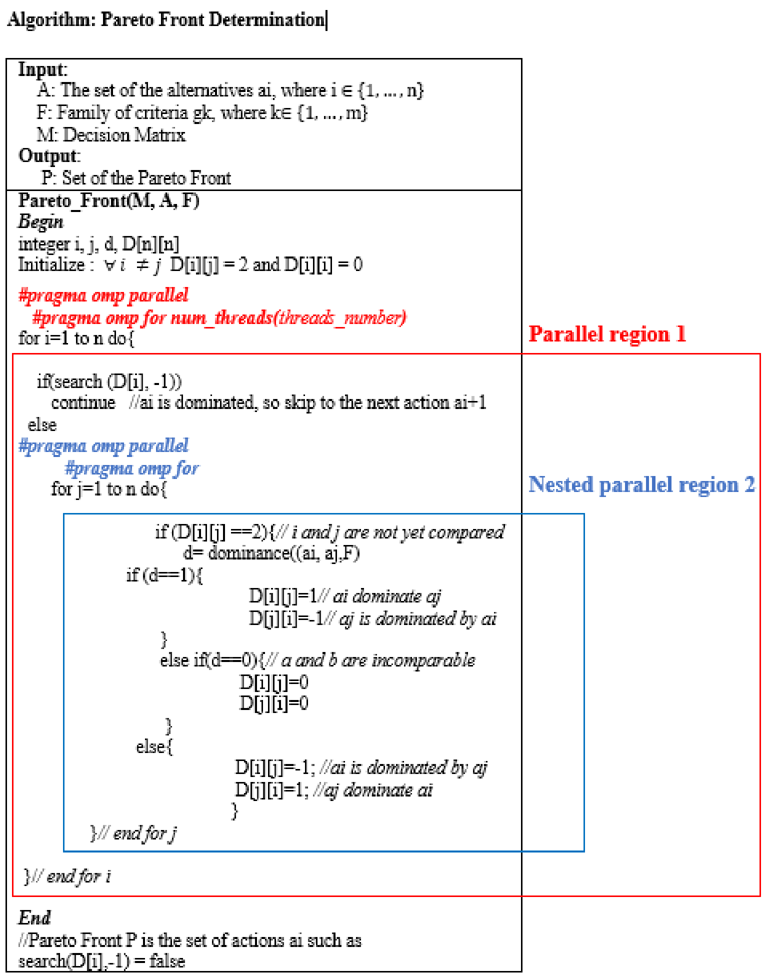
The Pareto filter proposed in [1] is a potential candidate for parallelization for different reasons. The large number of iterations required in the pairwise comparisons and the independence between the iterations which compare, for each pair belonging to the set of alternatives, the group of criteria taken into consideration. The idea is to execute in parallel several pairwise comparisons. We choose to parallel the section that consumes the most execution time in the sequential algorithm and overcome communication overhead by using the Open MP directives.

Large Open MP parallel regions are used because fragmented parallel regions would increase the overhead of creating and terminating threads. The proposed program begins as a single execution thread called the master thread. When the thread meets the parallel sections, it creates a team of threads consisting of the initial thread itself and the other threads becomes the team's main thread. All the members of the team collaborate to update the dominance matrix. At the end of the parallel sections, the further execution of the code is performed only by the master thread.

The parallel program is written in a way that facilitates the automatic parallelization over no-data-dependency loops by compile directives. The pragma OMP PARALLEL is used to fork additional threads to carry out the work.

For example, the proposed algorithm in [1] contains two nested regions (see blow: loops framed in red and blue) which are candidates for parallel execution. On these two regions, we have included the **pragma omp parallel for** and indicated the number of threads to use:

#pragma omp parallel for $num\_threads\ (number\ of\ threads\ to\ use)$



Algorithm: Pareto Front Determination

```
Input:
  A: The set of the alternatives ai, where i ∈ {1, ..., n}
  F: Family of criteria gk, where k∈ {1, ..., m}
  M: Decision Matrix
Output:
  P: Set of the Pareto Front

Pareto_Front(M, A, F)
Begin
integer i, j, d, D[n][n]
Initialize : ∀ i ≠ j  D[i][j] = 2 and D[i][i] = 0
#pragma omp parallel
  #pragma omp for num_threads(threads_number)
for i=1 to n do{                                          Parallel region 1
   if(search (D[i], -1))
     continue   //ai is dominated, so skip to the next action ai+1
   else
#pragma omp parallel
      #pragma omp for
     for j=1 to n do{                                     Nested parallel region 2
          if (D[i][j] ==2){// i and j are not yet compared
             d= dominance((ai, aj,F)
          if (d==1){
                D[i][j]=1// ai dominate aj
                D[j][i]=-1// aj is dominated by ai
          }
          else if(d==0){// a and b are incomparable
                D[i][j]=0
                D[j][i]=0
          }
          else{
                D[i][j]=-1; //ai is dominated by aj
                D[j][i]=1; //aj dominate ai
                }
     }// end for j
}// end for i
End
//Pareto Front P is the set of actions ai such as
search(D[i],-1) = false
```

5. Experiments and evaluation

In this section, we assess the performance of the parallel implementation of the filter with Open MP. The platform for conducting the experiments is an IntelCore i7 2.60 GHz laptop with 4 cores, 8 logical cores. The program was implemented using C language with Open MP 5.1. We conducted several experiments to measure the advantages of using Open MP. Our empirical study focused on the following parameters:

- The problem dimension: the number of alternatives
- The number of threads used.

The decision matrix used is based on random numbers with a high number of alternatives (200 to 20.000) and 10 criteria. We have collected several measures to evaluate the performance of the filter :

1. The execution time $T(p)$: depends on the size of the problem and the number of utilized threads p.
2. The speedup $S(p)$: indicates the degree of performance obtained. It presents the ratio between the sequential program execution times and the parallel program using p threads.

$S(p) = \frac{T(1)}{T(p)}$ with : $T(1)$ is the execution time using the sequential program

The computation time according to the number of alternatives and the number of threads is summarized in table 2 .
The results show that parallel computing can speed up computation mainly for a large number of alternatives. But it runs at its maximum efficiency before an overhead saturation, reached when handling a large number of threads takes too long and slows down improvement in execution time, see figure 4.

Table 2. Computation time (in ms) according to the number of alternatives and threads

alternatives	sequential	4 threads	8 threads	12 threads	16 threads	20 threads	24 threads
200	2	0,8	0,1	0,01	0,04	1	7
500	7,998	3,18	2,03	1,82	0,001	1	8
1000	23	10,2	5,26	3,67	2,5	7	8
1500	50,99	19,99	12,05	8,27	7,14	8	8
2000	88	37	20	15	12	16	8
5000	522	225	155	96	70	62	63
10000	2137	965	538	373	282	222	1019
20000	9672	4034	2135	1647	1449	184	1069

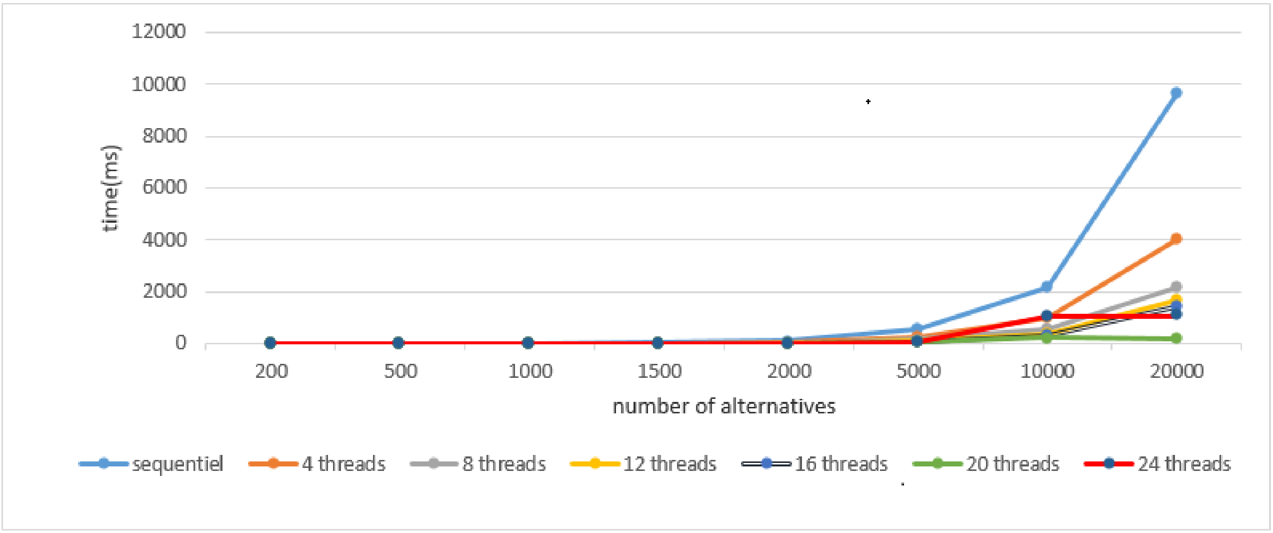


Figure 4. Variation of the computation time according to the number of alternatives in the decision matrix and the number of threads

The curve in figure 5 shows the variation of the computation time according to the number of threads. There are three different parts on the graph:

- For 1 to 8 threads, the gain in computing time is very close to the theoretical gain, the parallelism is excellent. The threads are all processed by processor cores. The system behaves like a computer with 8 processors.

- Between 9 and 12 threads, the curve no longer follows the same performance. The increase in the number of threads is still interesting. From 9 threads, hyperthreading technology is used to manage additional threads.

- Beyond 12 threads, the result is stable. Managing threads takes too long and slows down improvement in execution time.

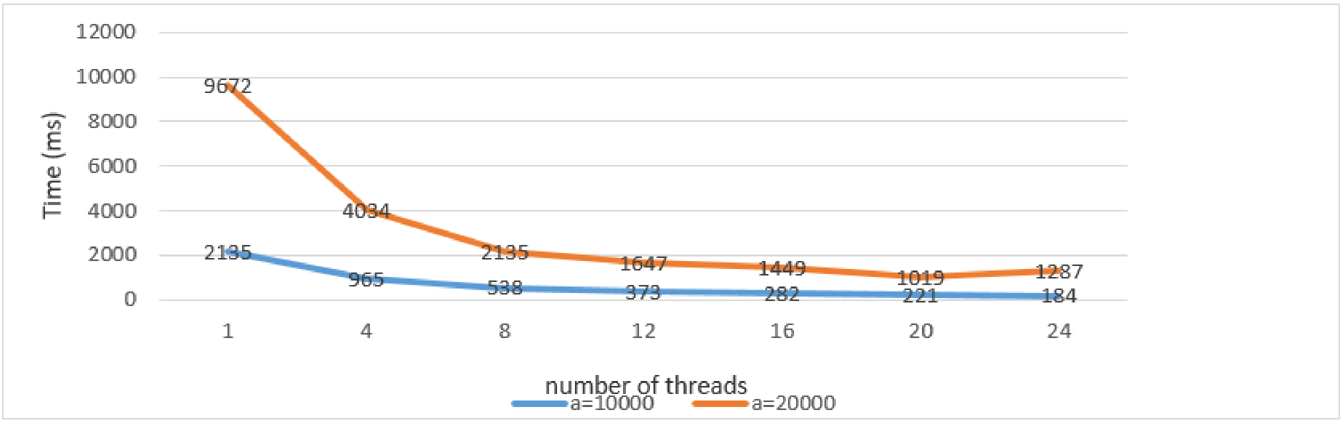


Figure 5. Computation time according to the number of threads

The curve in figure 6 shows the variation of speedup achieved with 20000 alternatives according to the number of threads used. A green line represents the theoretical speedup.

The speedup roughly follows the theoretical curve, but it starts to deteriorate from the use of 20 threads (thread overhead). Adding more threads usually helps, but after some point, they cause some performance degradation. Finding the right number of threads depends on the decision matrix size and the architecture it runs on.

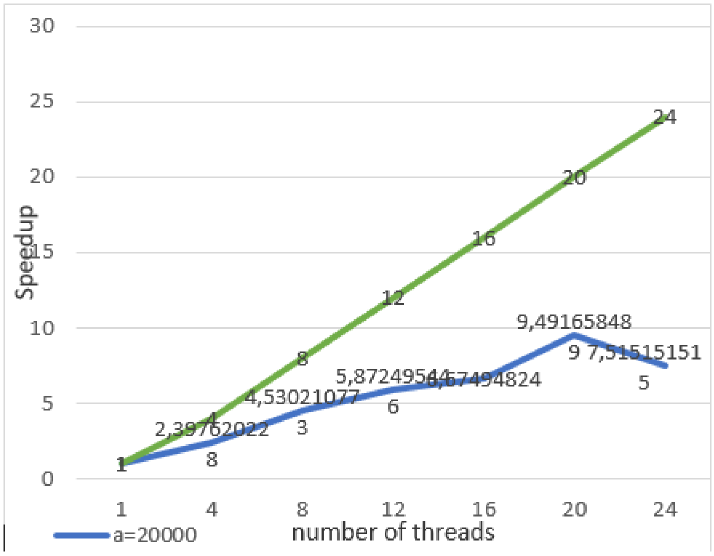


Figure 6. Speedup curves with 20000 alternatives

Conclusion

In this paper, we have proposed a parallel approach based on Open MP to filter the Pareto front from a large set of alternatives. The nature of the program and the features offered by Open MP made parallelization more accessible and straightforward. At the same time, some modifications had to be made to the algorithm to obtain the level of efficiency that we achieved. The proposed implementation performs very high even with the increased size of problem dimensionality. It can be widely used in real-world decision-making cases. It can be incorporated into the analysis of robustness in the Multi-criteria Decision Making process to generate results in a reasonable time.

High-performance computing-based parallel computing is a very promising technology to speed up the computation and facilitate the decision process. On the horizon, there are radically new solutions such as quantum computing, optical computing, which all possess the potential for future parallel computing. All this progress is tightly connected with the development of parallelization models and algorithms for these systems. Without effort in this field, the computational power of super-modern computers, which are today available, cannot be exploited.

in our future work, we want to investigate the possibility and gain of parallel implementation by using GPU accelerators in parallel with CPUs to speed up the decision process. And so, we can take advantage of technological advances in hardware and modern development paradigms, especially with the emergence of computers equipped with high-performance GPUs.

References

1. L. Lamrini, M. C. Abounaima, M. Talibi Alaoui, F. Z. El Mazouri, N. El Makhfi, and M. Ouzarf, “A filtering approach used in a massive data context to reduce the set of choices in a multicriteria decision aid process: Pareto solutions,” Dec. 2020, doi: 10.1109/ICECOCS50124.2020.9314445.
2. M. Yasmin, E. Tatoglu, H. S. Kilic, S. Zaim, and D. Delen, “Big data analytics capabilities and firm performance: An integrated MCDM approach,” J. Bus. Res., vol. 114, no. August, pp. 1–15, 2020, doi: 10.1016/j.jbusres.2020.03.028.
3. B. Yatsalo, V. Didenko, S. Gritsyuk, and T. Sullivan, “Decerns: A Framework for Multi-Criteria Decision Analysis,” Int. J. Comput. Intell. Syst., vol. 8, no. 3, pp. 467–489, May 2015, doi: 10.1080/18756891.2015.1023586.
4. V. Yadav, S. Karmakar, P. P. Kalbar, and A. K. Dikshit, “PyTOPS: A Python based tool for TOPSIS,” SoftwareX, vol. 9, pp. 217–222, Jan. 2019, doi: 10.1016/j.softx.2019.02.004.
5. L. C. Dias, J. P. Costa, and J. N. Clímaco, “Conflicting criteria, cooperating processors-some experiments on implementing a multicriteria decision support method on a parallel computer,” Comput. Oper. Res., vol. 24, no. 9, pp. 805–817, 1997, doi: 10.1016/S0305-0548(97)00001-4.
6. M. Aruldoss, T. M. Lakshmi, and V. Prasanna Venkatesan, “A Survey on Multi Criteria Decision Making Methods and Its Applications,” Am. J. Inf. Syst., vol. 1, no. 1, pp. 31–43, 2013, doi: 10.12691/ajis-1-1-5.
7. K. Govindan and M. B. Jepsen, “ELECTRE: A comprehensive literature review on methodologies and applications,” European Journal of Operational Research, vol. 250, no. 1. Elsevier, pp. 1–29, Apr. 01, 2016, doi: 10.1016/j.ejor.2015.07.019.
8. M. Behzadian, R. B. Kazemzadeh, A. Albadvi, and M. Aghdasi, “PROMETHEE: A comprehensive literature review on methodologies and applications,” Eur. J. Oper. Res., vol. 200, no. 1, pp. 198–215, Jan. 2010, doi: 10.1016/j.ejor.2009.01.021.
9. G. Schryen, “Parallel computational optimization in operations research: A new integrative framework, literature review and research directions,” European Journal of Operational Research, vol. 287, no. 1. Elsevier B.V., pp. 1–18, Nov. 16, 2020, doi: 10.1016/j.ejor.2019.11.033.
10. L. C. Dias, J. P. Costa, and J. N. Clímaco, “Conflicting criteria, cooperating processors-some experiments on implementing a multicriteria decision support method on a parallel computer,” Comput. Oper. Res., vol. 24, no. 9, pp. 805–817, Sep. 1997, doi: 10.1016/S0305-0548(97)00001-4.
11. L. Dias, J. P. Costa, and J. N. Clímaco, “A parallel approach to the analytic hierarchy process decision support tool,” Comput. Syst. Eng., vol. 6, no. 4–5, pp. 431–436, Aug. 1995, doi: 10.1016/0956-0521(95)00045-3.
12. L. C. Dias, J. P. Costa, and J. N. Clímaco, “A parallel implementation of the PROMETHEE method,” Eur. J. Oper. Res., vol. 104, no. 3, pp. 521–531, Feb. 1998, doi: 10.1016/S0377-2217(97)00007-6.
13. P. Bauchet, “Pareto (Vilfredo) - Cours d’économie politique.,” Rev. économique, vol. 16, no. 5, pp. 811–812, 1965.
14. S. E. Thompson and S. Parthasarathy, “Moore’s law: the future of Si microelectronics,” Mater. Today, vol. 9, no. 6, pp. 20–25, Jun. 2006, doi: 10.1016/S1369-7021(06)71539-5.
15. “54 September 2005 QUEUE.” Accessed: Mar. 26, 2021. [Online]. Available: www.acmqueue.com.
16. D. Etiemble, “45-year CPU evolution: one law and two equations,” arXiv, 2018.
17. M. Schmeisser et al., “Parallel, distributed and GPU computing technologies in single-particle electron microscopy,” Acta Crystallogr. Sect. D Biol. Crystallogr., vol. 65, no. 7, pp. 659–671, 2009, doi: 10.1107/S0907444909011433.
18. M. C. Gabriella Cattaneo, Chris Ingle, S. Stefania Aguzzi, and V. S. Conway, Mario Morales, The Impact of the Introduction of Multicore Technologies on the Computing Market and Opportunities for Europe. 2013.
19. F. P. Violaine Louvet, “Introduction au calcul parallèle - Contexte et motivations,” 2021.
20. K. S. Galtin and P. Isensee, “OpenMP and C++: Reap the Benefits of Multithreading without All the ...,” October, vol. 163717, pp. 1–14, 2010, Accessed: Feb. 26, 2021.

21. “32 OpenMP traps for C++ developers,” 2020.
https://software.intel.com/content/www/us/en/develop/articles/32-openmp-traps-for-c-developers.html (accessed Feb. 22, 2021).
22. H. Date and H.- Hand, “An introduction to Parallel Programming with MPI and OpenMP,” no. August, pp. 1–12, 2013.