International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 10 Issue 7 ǁ July 2022 ǁ PP. 104-106

A Comparative Study of Performance Using Sequential Vs Parallel Programming Applied In Genetic Algorithms

Eldaief Donelles[1], Henriqe A. Richter[1], Miquéias F.M. Trennepohl[1], Taís T. Siqueira[1], Rogério SM Martins[1], Carlos D. Tweneboah[2]
[1]*Department of Computer Science, Regional University of the Northwest of the State of Rio Grande do Sul, Rio Grande do Sul, Brazil*
[2]*Department of Computer Engineering, University of Lagos, Lagos, Nigeria*

Abstract
This work presents a comparative study of performance between 3 different implementation methods, which are applied to the same problem. Initially, a sequential implementation was developed that served as the basis for the following 2, which made use of the OpenMp and POSIX Thread libraries for parallelization of their algorithms. An interesting fact to be highlighted, and which may explain the low performance of the Pthread library is that, when executing the sequential algorithm, it used 100% one core, OpenMP used close to 80% to each core, whereas Pthread used around 60% of each allocated core.

Keywords: *OpenMp, POSIX, nqueens, parallel computing, performance evaluation.*

Date of Submission: 24-06-2022 Date of acceptance: 05-07-2022

I. INTRODUCTION

With the increasing dissemination of multi-core computational architectures, which provide the processing of several tasks in parallel, thus generating significant performance increases, there has been a massive increase in research aimed at exploiting these resources, due to the fact that in many cases, part of this available processing power is not used in application development. It is in this context that the optimization of algorithms is inserted, which makes use of some parallelization methods, among which, we can mention the use of OpenMP and POSIX Threads libraries.

This work presents a comparative study of performance between 3 different implementation methods, which are applied to the same problem. Initially, a sequential implementation was developed that served as the basis for the following 2, which made use of the OpenMp and POSIX Thread libraries for parallelization of their algorithms.

To carry out the different implementations, the Genetic Algorithm meta-heuristic was chosen, which seeks a global optimization and is based on the mechanisms of natural selection and genetics. Basically, a population with n individuals is generated, each individual containing information that generates a particular solution for the analyzed problem. As long as an individual is not found that contains a satisfactory solution, the crossing between the individuals is carried out, generating a new population, thus starting a new iteration in search of the defined solution, according to Linden (2012).

The problem chosen for the parallelization of the algorithms was the nqueens problem, this is an exponential combinatorial problem that consists of placing nqueens on a chessboard with size nxn so that there are no collisions, according to Laguna (1994). A collision occurs when two or more queens are positioned in the same row, column, or diagonal. However, it is not the objective of this work to find the solution to this problem, but to measure and analyze the performance of the 3 algorithms implemented, considering the same number of iterations.

II. METHODOLOGY

In the process of studying and developing the topic addressed in this work, theoretical knowledge was sought on: the nqueens problem, the genetic algorithm meta-heuristic, the two parallel programming methods (OpenMP and Pthread) and the C++ programming language, in order to acquire understanding enough to develop the implementations. In the following paragraphs some concepts about threads are presented and the parallel programming methods used in this work are briefly explained.

Thread is a way for a process to divide itself into several tasks that can run in parallel, making the system manage each task independently. Unlike processes, which are independent and memory is not shared, threads in the same process can use the same instructions and variables. In addition to taking up less memory, context switching between threads is faster than between processes, according to Edward (2006).

Therefore, threads are important elements in parallel processing, because through their use it is possible to make a sequential code (executed by only one thread) be divided between several threads and have its performance improved. To accomplish this division of processing between multiple threads, programmers rely on the help of libraries that help in the parallelization of the sequential codes developed. Among the libraries offered, PThread and OpenMP can be highlighted, which are described below.

Pthread are libraries for languages That C++allow generating a set of threads that can run simultaneously. Implementations made with these libraries are more efficient in multi-processor and/or multi-core systems where the created threads will be executed in multiple processors/cores, allowing the program execution time to decrease, gaining more speed through parallel processing, this higher performance can also be achieved on single-processor systems. YOLINUX (2014).

The Pthread library works with parallel programming with explicit threading, that is, the programmer is responsible for explicitly creating the multiple threads and defining the work that will be carried out by them. In addition to creating the threads, the programmer must also manage the synchronization between them, so that the parallelization of the sequential code does not corrupt the results obtained.

So that the Pthread library can be used in the programming language C++, which was used to develop the application in this work, it is necessary to import it through the command "#include <pthread.h >". After performing the import, it is now possible to use the resources and functions provided by the Pthread library, such as creating a new thread using the "pthread_create" command and synchronizing it using the "pthread_join" command.

OpenMP is a standard used for parallelizing applications. This standard was developed and is maintained by the OpenMP Architecture Review Board group, which is formed by companies with large participation in software and hardware development, such as IBM, Intel, SUN, among others. It arose from the need that these large companies found to formulate a standard for parallel programming in environments with shared memory. In these environments, the various existing processing cores make use of the same memory resource, so there is no need to exchange information between the processing cores, however, it is necessary to synchronize them so that problems do not occur. access to shared memory, according to CALDAS and SENA (2008).

Contrary to what some people may think, OpenMP is not a programming language, but a standard that defines an API for parallel programming in shared memory. It gathers a set of compilation directives, routines and environment variables. The compilation directives of the OpenMP standard can be defined as different lines of code, which have the objective of informing the compiler how to parallelize a given piece of code.

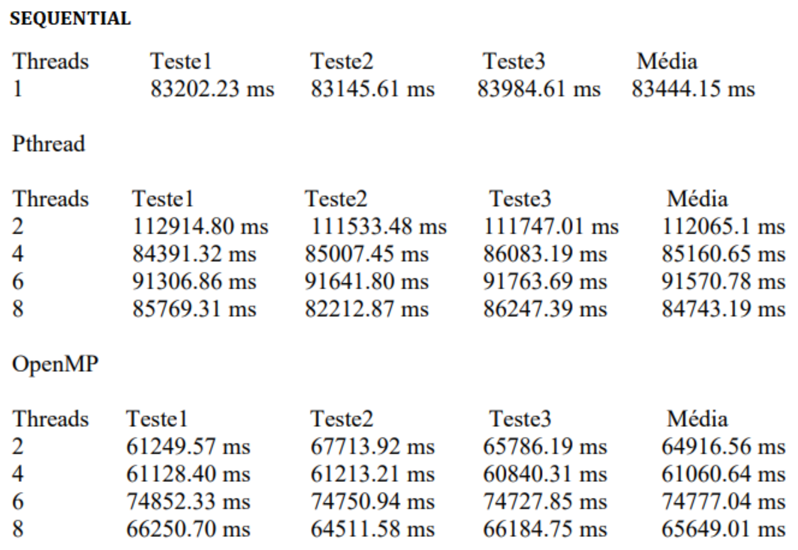
In the C++ programming language, which was the language used to develop the application presented in this work, parallelization through OpenMP occurs through the use of the compilation directive #pragma omp. This directive must be defined just above the code to be parallelized.

So that the C++ compiler can correctly perform the parallelization of codes marked with the OpenMP compilation directives, the -fopenmp option must be added to its execution command. In addition, it is necessary to import the library "omp.h" so that it is possible to use the functions and compilation directives existing in the OpenMP API.

The parallelization through OpenMP was used in this work through the compilation directive "#pragma omp parallel for" to parallelize the loop of repetition responsible for the evaluation of the population generated through the genetic algorithm used.

III. RESULTS AND DISCUSSION

To carry out the tests and collect results for the case study discussed in this work, the following were used: a board 128×128 (capacity for 128 queens), population with 120 individuals and defined 5000 iterations. Below is the display of the results obtained.



SEQUENTIAL

Threads	Teste1	Teste2	Teste3	Média
1	83202.23 ms	83145.61 ms	83984.61 ms	83444.15 ms

Pthread

Threads	Teste1	Teste2	Teste3	Média
2	112914.80 ms	111533.48 ms	111747.01 ms	112065.1 ms
4	84391.32 ms	85007.45 ms	86083.19 ms	85160.65 ms
6	91306.86 ms	91641.80 ms	91763.69 ms	91570.78 ms
8	85769.31 ms	82212.87 ms	86247.39 ms	84743.19 ms

OpenMP

Threads	Teste1	Teste2	Teste3	Média
2	61249.57 ms	67713.92 ms	65786.19 ms	64916.56 ms
4	61128.40 ms	61213.21 ms	60840.31 ms	61060.64 ms
6	74852.33 ms	74750.94 ms	74727.85 ms	74777.04 ms
8	66250.70 ms	64511.58 ms	66184.75 ms	65649.01 ms

By analyzing the data extracted through the tests carried out, it can be seen that by using the OpenMP library, it was possible to achieve the best performance results for the application. The sequential implementation had the second best performance and the implementation using the Pthread library had the worst performance. An interesting fact to be highlighted, and which may explain the low performance of the Pthread library is that, when executing the sequential algorithm, it used 100%one core, OpenMP used close to 80% to each core, whereas Pthread used around 60% of each allocated core.

IV. CONCLUSION

When completing the work and analyzing the results of the tests that were carried out, one can see the superior performance capacity of the algorithm that was implemented using the OpenMP library, although both parallel algorithms can be further improved in order to optimize them for a certain multi-core architecture, and perhaps this way they can reach similar results. It can also be seen in the present work, how much the libraries studied here help in the practice of parallel programming, especially OpenMP, which is responsible for most of the parallel programming, by simply defining the code snippet that the same want to parallelize that the API takes care of creating the threads and synchronizing them.

REFERENCES

[1]. Caldas, José A. and SENA, Maria C. Tutorial OpenMp C/C++. Electronic analyses. [Maceió, AL], 2008. Available at: < http://www.lbd.dcc.ufmg.br/colecoes/erad-rs/2010/006.pdf >. Accessed on: May 15, 2022.
[2]. Edward A. Lee, The Problem with Threads. Electronic analyses. [Berkeley, CA], 2006. Available at: < http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.pdf >. Accessed on: Jun 06, 2022
[3]. Laguna, M. A Guide to Implementing Tabu Search. Investigacion Operativa, 4:159-178,1994.
[4]. Linden, Ricardo. Genetic Algorithms. 3rd ed. modern science,2012.
[5]. Yolinux. Site. POSIX Thread (pthread) libraries. Available at: http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html . Accessed on: Jun 06, 2022.
[6]. Dutta, Sourav, Sheheeda Manakkadu, and Dimitri Kagaris. "Classifying performance bottlenecks in multi-threaded applications." In *2014 IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs*, pp. 341-345. IEEE, 2014.
[7]. Manakkadu, Sheheeda, and Sourav Dutta. "Bandwidth based performance optimization of Multi-threaded applications." In *2014 Sixth International Symposium on Parallel Architectures, Algorithms and Programming*, pp. 118-122. IEEE, 2014.
[8]. Sri Vidya, B., Harini Sriraman, and P. Rukmani. "Optimized Multi-threading To Balance Energy and Performance Efficiency."
[9]. Sriraman, Harini, and Pattabiraman Venkatasubbu. "Extending Lifetime Reliability Model for Multi-Threaded Architectures." *International Journal of Next-Generation Computing* (2018): 51-65.
[10]. Sriraman, Harini, and Pattabiraman Venkatasubbu. "SeRA: Self-Repairing Architecture for Dark Silicon Era." *Journal of Circuits, Systems and Computers* 29, no. 04 (2020): 2050053.
[11]. Raj, Supragya, Siddha Prabhu Chodnekar, T. Harish, and Harini Sriraman. "eMDPM: Efficient Multidimensional Pattern Matching Algorithm for GPU." In *Smart Innovations in Communication and Computational Sciences*, pp. 97-104. Springer, Singapore, 2019.