Realization of Axon II Chess Engine

Vladan Vučković
Computer Department
University of Niš, Faculty of Electronic Engineering

Niš, Serbia
vladanvuckovic24@gmail.com, ORCID 0000-0002-7552-9596

***Abstract*—This paper will present the details of the construction of the author's new chess program Axon II (version 2026). The combinatorial explosion problem, which is fundamental to all applications for intensive decision tree processing, is addressed in the new engine by a combination of modern tree processing techniques and intensive use of X86 machine code for programming the basic functions of the engine, primarily the position generator and evaluator. Empirical tests on various platforms show very high raw engine speed factor (NPS factor) in comparison with referent chess engine - Stockfish 18.**

Keywords— Theory of Logic Games, Computer Chess, Chess Engines, Machine Programming.

I. Introduction

The experimental chess program Axon I, which the author has been developing for several years and which in its latest version has reached the chess strength of strong chess master (ELO rating 2500), is the basis for the development of the program Axon II 2026. The basic construction details of new engine are presented in this paper. Since 2001, the program Axon, in different versions, has successfully participated in a number of tournaments, competing on an equal footing with chess masters. The parallel version of the chess system Achilles, which was developed on the basis of research into advanced algorithms in the field of parallelization, shows exceptionally good results in the game against other programs. Also, using Internet technologies, the parallel system Achilles is connected to the world's largest chess server Chess Base.

In contrast to the approach applied in the program Axon I, whose work is based on basic positional matrices of dimensions 8x8 bytes, Axon II switches completely to 32-bit data organization. Each field in the matrix representing the chessboard is 32-bit, and contains complete information that is essential for the operation of the power generator and evaluator. Unlike the dominant *bit board* structure that is now almost completely used in modern PC chess machines, including Stockfish 18, Axon II has a completely different and original approach. Instead of clusters of *bit boards* that must be constantly transferred through the decision tree, only a single structure is used - a compact 32-bit matrix. The machine code is also simplified and does not require 64-bit machine instructions, but only 32-bit ones. Given this consistency between code and data structure, it turns out that the practical realization of this chess machine itself has remarkable performances, as shown by comparative empirical analyses.

Of course, Axon II is only an experimental basis for a future chess engine and needs to be greatly expanded to achieve a master level of play, but it shows the potential of a new and original approach. Modern superscalar processors are extremely optimized for executing this instruction stream, so that machine code written by hand can reach very high speeds - faster than currently available PC chess engines. Empirical research is focused exclusively on performance on a single processor thread (processor core), so that we can count on multiplying performance in future versions that will use multiprocessor techniques.

It is important to emphasize that the focus of this paper is exclusively on comparing the engine raw power - speed of the position calculation of the basic core engine, which is manifested in the NPS factor. As a reference engine for measuring speed, Stockfish 18 was chosen, which is the latest version, open-source and freeware engine and that be easily downloaded and checked. Let us emphasize that the subject of the paper is not measuring the total chess power of both engines, since it depends on many other factors, besides the speed of position calculation. Stockfish 18 is certainly the leading world PC engine in terms of total chess power at the moment.

II. Definition of Chess as a Game

First, we will formally define chess as an instance of game theory, which allows the application of the laws and rules of game theory to its study [1], [2]. In this way, a scientific basis has been formed for the implementation of chess algorithms that create a computer chess program, representing the formalization of machine thinking (artificial intelligence) in the field of game theory.

a) Chess is a strategic game in which two players participate, whose moves alternate. The players are designated by the terms white and black. It is understood that the players play rationally.

b) Chess is a game with perfect information and zero sum (zero-sum perfect information game).

c) Chess is a combinational game of the partisan type. The number of possible legal moves in each position is finite and can be different for each player.

d) Chess has three possible values of the payoff function: a win for White, a win for Black, and a draw. The payoff function in chess can be defined by the simple payoff matrix

e) A chess game always ends in a finite number of moves regardless of the players' strategies.

The basic minimax theorem can be applied to chess: *For any finite game with fully defined information, involving two*

ISBN 978-86-6200-074-3©2026AM& ETRAN
https://doi.org/10.69994/13Ic26040

players and having a zero sum, an optimal strategy for the game can be defined.

Under the given conditions, the following equality can be defined:

$$\max_X \min_Y X^T AY = \min_Y \max_X X^T AY = V$$

Where:

V - game value (evaluation), A - payoff matrix ,

X,Y - solution matrix.

Logically, this theorem defines the procedure for finding the best continuation for player A, which at the same time represents the maximum value of the possible continuations available to him. Player B, playing against player A, tries to maximize his payoff function, or to minimize the payoff function of player A. The presented definitions and the minimax theorem represent the basis for the development of algorithms for treating logical games in the field of artificial intelligence, as well as for the creation of computer programs that play chess [2].

III. THE CONCEPT OF COMPLEXITY AND COMBINATORIAL EXPLOSION

In a real chess game, the roles of players A and B alternate, so that in the presented formula, the minimization and maximization procedures are constantly alternating. The application of the formula is actually recursive. Determining the correct strategy for the player whose turn it is is based on an extension in the form of a tree. At each node in the tree, a procedure of maximization or minimization of values is performed. If we assume that each node has 30 outputs, it is clear that the number of nodes is exponentially dependent on the depth d, so that the total number of nodes in the tree is of order 30^d. Thus, from the field of game theory, the problem of complexity appears in the form of an extreme increase in the number of nodes in the tree, which is also a synonym for the concept of combinatorial explosion, which is known in the field of artificial intelligence. Since the depth of the minimax tree directly affects the chess strength of a program, it is clear that the complexity problem must be solved satisfactorily in order to obtain a rational method for creating chess programs.

A. Methods For Overcoming The Complexity Problem

It is clear that the complexity problem cannot be overcome by brute force, i.e. by calculating all possible nodes when solving the minimax formula [2],[3]. Given the limited capabilities of hardware, the problem of overcoming complexity is approached in two ways: by the heuristic tree pruning method (forward pruning) [3]. or by the numerical pruning method (backward pruning). These two strategies were described by Shannon in his original work and called them the type A strategy and the type B strategy.

B. Heuristic Pruning Of Decision Trees (Forward Pruning)

The heuristic pruning method originates from an analogy with the way of thinking of chess masters [1],[3]. It is obvious that people do not have a high computational speed, but they can achieve an extremely high level of play. Also, by studying the way of thinking of chess masters, one comes to the conclusion that the list of variants considered by chess players is extremely narrow and that some variants are calculated to great depths while other lines are ignored [3]. Therefore, the application of heuristic, expert knowledge in the phase of generating extensions from each node is primary. If we assume that we have a function that generates only the 3 best extensions in each node, which are considered further, it is clear that the number of nodes in the tree is 3n. If we then assume that 30 moves are considered at each node, a tree of depth 5 will require 24.3 million nodes. For 3 outputs per node at depth 15, only 14.3 million nodes will be needed. It can be concluded that with 50% fewer nodes, a 3-fold greater computational depth is achieved.

The basic assumption of this type of tree pruning is the existence of a stable heuristic that will consider the really important moves [1],[3]. The biggest problem, which has never been completely solved, is the heuristic of rejecting continuations that are considered irrelevant [3]. In practical play, rejection of important continuations and major oversights in the calculation often occur. The problem of successful heuristics in this case has not been solved satisfactorily to date.

C. The Method Of Numerical Pruning Of The Decision Tree (Backward Pruning)

The second method assumes that it is not necessary to develop special heuristics, but it is necessary to maximally speed up all functions and consider all extensions [4]. This type of heuristic is actually a brute force method where the algorithm goes through all legal extensions in the minimax tree. The heuristics related to the chess game itself are implemented in the evaluation function that is calculated for each terminal node [6]. Given that minimax values are available at various stages of the game, a new concept of tree pruning is introduced, called numerical pruning or backward pruning. This method includes the basic ALPHA-BETA algorithm [3]. Unlike heuristic pruning, which is theoretically uncertain, numerical ALPHA-BETA pruning, which is based on the backward propagation of numerical values of the evaluation of terminal nodes, is absolutely certain [6]. In all possible tree variants, the ALPHA-BETA algorithm gives the same best moves as the basic minimax algorithm, so in computer chess this method is used as the basis of almost all programs.

D. Limiting Tree Expansion

As we have pointed out, the combinatorial explosion problem significantly limits the possibility of processing the tree to greater depths. On the other hand, limiting the depth of computation in all variants is the basic technique for limiting tree expansion. The effect of tree depth on game strength has been studied in detail in the works of Schaeffer, who came to the conclusion that the same program that computes to a fixed depth D wins 80% of the victories compared to a program that computes to a depth D-1 [2]. According to this law, the strength of a chess program grows almost linearly with depth, at a rate of about 200 ELO points per half-move. According to Ken Thomson's law, in order to beat the world chess champion, it is necessary to process the tree to a depth of 14 half-moves in each move. This law is easy to explain if we consider that Garry Kasparov had a strength of about 2800 ELO points when he was the world chess champion [3].

Therefore, limiting the depth as the main method of implementing tree processing in finite time conditions and preventing combinatorial explosion is counterproductive in

terms of chess strength. Applying the depth limiting technique leads to the appearance of another negative effect, which is called the horizon effect [1],[2],[3]. If the depth of the tree is directly limited, a large instability in the evaluation occurs at the level of terminal nodes. An artificial depth limit can interrupt a series of figure exchanges or generate some other form of dynamic instability. Therefore, instead of evaluating terminal nodes at these points, calls to a special procedure are used whose task is to reduce dynamically active positions to their static equivalents. The use of additional procedures at the level of terminal positions leads to a dynamic change in depth as a function of the instability of the node being evaluated. In this way, the optimal structure of the tree is achieved: at the basic level Shannon type A processing method, and at the level of terminal nodes Shannon type B selective processing of the tree (quiescence search) [1].

IV. Standalone Chess Machines - Axon I

The principle of standalone chess machines involves the integration of hardware and software in the form of a compact device. Moves are transmitted to the processor via a sensor chessboard on the upper part of the device. Devices of this type were once very popular due to their mobility, but today, with the advent of portable PCs, they are losing their importance.

With the advent of PCs, whose speed was developing explosively, the power of programs based on these machines grew rapidly. Initially, programmers developed integrated systems that contained both a chess graphical environment and a chess program (chess engine) in the same executable file. The author's early projects, the Geniss Axon and Axon programs, also belong to this type (Fig. 1) [3],[4],[5].

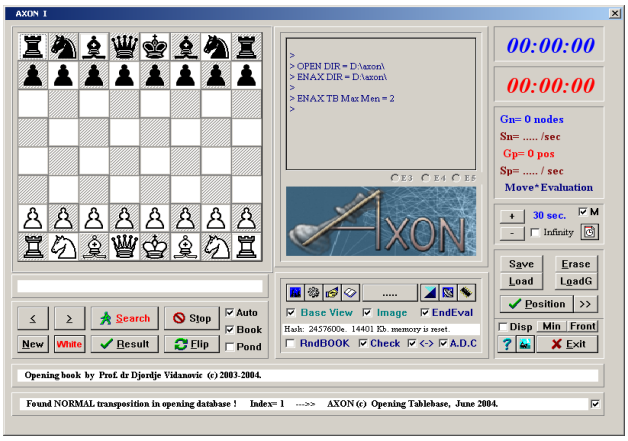

Fig. 1. Integrated Axon I chess system – graphical environment and program.

The advantages of integration are numerous, because there is a direct memory connection between the two systems, so synchronization problems do not exist. Setting the options of chess programs directly from the program environment is straightforward. The main disadvantage of this approach is the need to simultaneously develop both the graphical environment and the program. On the other hand, playing automatic games against other programs, which are most important in terms of objectively assessing the strength of the program, is very difficult to implement. The principles of connecting two computers via a communication cable for this purpose have almost been abandoned. Regardless of the fact that this approach was very popular in the beginning even with top professional companies, it has been abandoned, so that systems in which the graphical environment is separated from the chess program itself are now dominant.

V. Axon II Engine Construction

In this section, step by step, the principles of construction and the basic elements of the structure of a novel chess program Axon II will be presented. As already emphasized, a chess program is a very complex application that contains implemented numerous techniques from various fields of programming. The theoretical basis for building programs that play at the level of strong chess masters stems from the basic minimax decision-making principle, which has its analogy in the field of normal decision-making in various forms of strategic games. In short, if there are two parties playing a logical game, and this is exactly the situation in a chess game, each of them tries to choose the best of all the moves available to it. On the other hand, the opponent's strategy is to choose the best move for itself. Since the interests of the opposing parties are in principle directly opposed, the party that has the move has the task of choosing the best move not only in function of its position, but also in the perspective of the responses that the opponent can play. Given these facts, the so-called minimax principle of inference is formed, which represents the first and basic postulate of all chess and other programs that process logical games involving two parties. By applying this principle, a game tree is formed, which in theory is actually a decision tree. Ending - the basic elements of the tree are terminal nodes whose numerical value is calculated using an evaluation function. Using the minimax principle, the values of the best evaluations of the nodes of both sides are prolonged through the tree towards the basic - source node (root node). The player that has the move, as the best continuation, chooses the move that leads to the terminal node with the highest possible absolute value of the terminal node, applying the minimax principle at each level of the tree. As we have already mentioned, the basic theoretical and practical problem that needs to be solved by applying this principle is overcoming the problem of combinatorial explosion.

A. Basic Axon II Characteristics

Axon II v50 is the most powerful version of the program from the group of 32-bit applications. The graphical environment is integrated with the machine program. The program part implements standard and a number of advanced techniques. The core of the program is the ALPHA-BETA/PVS/R=2 null-move searcher, with a machine-defined generator and evaluation function. The v50 version is especially adapted for playing against people (“anti-grandmaster strategy”) by special adjustment of the evaluation function.

In addition to standard options, the program contains an implemented A.D.C. procedure, an UCI adapter for connection with other programs in the testing phase, a connection with the OPENX type transposition opening database and the possibility of connecting to another processor using the existing LAN network. Axon II contains a module for forming a dual system for parallel execution of programs on 2 processors according to the original split list method, the theoretical and practical basis of which is presented in the previous chapter. The power gain obtained in a dual processor configuration is about 40-50 ELO points.

B. *Axon II Engine Debugging and Test Utility*

For the purpose of developing and testing functions in the Axon II programming environment, a special application was developed that graphically represents an 8x8 matrix with an individual graphical representation of all 32 bits in each individual cell (Fig. 2) [5]. The structure of the figures shown corresponds to the basic arrangement of figures in Fig. 1. This approach to visualization greatly helps in the development of machine routines of the chess engine itself, considering that the action of individual machine instructions is immediately visible on the graphical interface. In this sense, debugging of the application is also greatly facilitated.

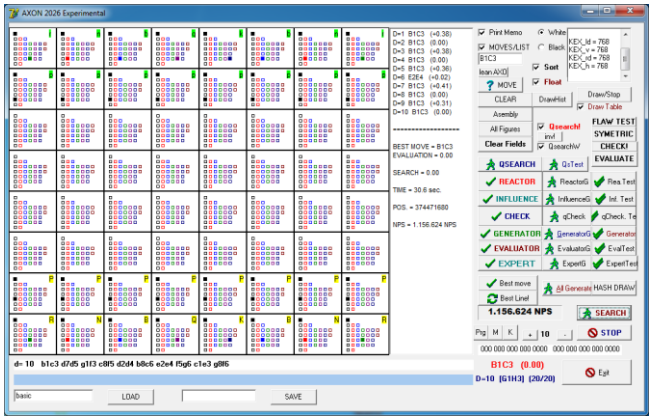

Fig. 2. Axon II engine developer and debugger.

The following image (Fig. 3) shows a group of four cells to illustrate the operation of this application. It is noticeable that the cell elements are designed to graphically display individual elements from a 32-bit matrix in different colors. In the author's experience, this way of presentation is incomparably more user-friendly for the programmer than classic debugging.

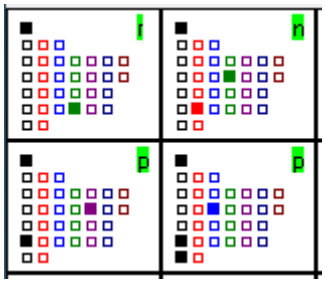

Fig. 3. Individual cells in Axon II engine visual debugger.

The structure of bits that directly correspond to the visual elements in a single cell is shown in the following list. The list shows that all qualitative elements, for both white and black, could be compressed into 32 bits. This contains all the information necessary for the operation of the generator and evaluator. This approach is excellent because it provides the possibility of further upgrading and using neural networks at the level of these basic data structures and procedures, which is the main trend in modern chess programs.

```
xfigure    = 1 shl 31;    {Piece bit definitions in 32 registar - for each cell}
{.}
xkingdiag  = 1 shl 30;    {Opposite king approaches - for check generator}
xkingvert  = 1 shl 29;
xkingknig  = 1 shl 28;
{.}
xwpawn     = 1 shl 27;    {Influence of white pieces}
xwknight   = 1 shl 26;
xwbishop   = 1 shl 25;
xwrookhorz = 1 shl 24;
xwrookvert = 1 shl 23;
xwqueen    = 1 shl 22;
xwking     = 1 shl 21;
{.}
xbpawn     = 1 shl 20;    {Influence of black pieces}
xbknight   = 1 shl 19;
xbbishop   = 1 shl 18;
xbrookhorz = 1 shl 17;
xbrookvert = 1 shl 16;
xbqueen    = 1 shl 15;
xbking     = 1 shl 14;
{.}
wpawn   = 1 shl 13;    {White pieces}
wknight = 1 shl 12;
wbishop = 1 shl 11;
wrook   = 1 shl 10;
wqueen  = 1 shl 9;
wking   = 1 shl 8;
{.}
bpawn   = 1 shl 7;     {Black pieces}
bknight = 1 shl 6;
bbishop = 1 shl 5;
brook   = 1 shl 4;
bqueen  = 1 shl 3;
bking   = 1 shl 2;
{.}
sensordiag = 1 shl 1;
sensorvert = 1;
```

This paper has been supported by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia [Grant Number: 451-03-34/2026-03/200102].

Given this bit arrangement, the basic structure (NUCLEUS) is defined, on which all basic functions of the chess engine work. This structure is very simple - it is an 8x8 matrix. The cardinal type in the Pascal programming language is an unsigned 32-bit integer:

NUCLEUS: ARRAY [0..7,0..7] of cardinal; {64*4 bit}

Here we can define the basic data structure used in the search engine. It contains all the information needed to run the minimax and alpha-beta algorithms.

```
TYPE
  stype = packed record
      NUCLEUS:ARRAY [0..7,0..7] of cardinal; {64*4}
      nucleusend:cardinal;
      WHITE_KING_YX:cardinal;   {King coordinates 0..63}
      BLACK_KING_YX:cardinal;   {y=byte   x=byte}
      all:cardinal;             {Total number of moves}
      newall:cardinal;
      flag_on_move:cardinal; {Flags- on move: WHITE/BLACK}
      last_move:cardinal;       {Last move complete}
      hash_high:cardinal;
      hash_low:cardinal;
      hash_addr:cardinal;
      piecesNUM:cardinal;       {Number of pieces}
      pieces:array [0..31] of cardinal;   {Piece  locations}
      Moves:array[0..127] of cardinal;  {Move list}
  end;
```

For demonstration, we will show an X86 machine routine that is one of the basic Axon II ones, in the part of scanning and processing the displayed matrix. The assembly routine has the task of scanning the matrix, finding the pieces and saving their characteristics on the stack. Since we have shown that our NUCLEUS matrix is the simplest data structure for representing a chessboard, it is clear that its processing by machine routines is also very efficient.

```
procedure SCANER_CORE;assembler;
{Register EAX is stack enter address}
{Register EDI is basic stack address}
asm
   push eax
   push ebx
   push ecx
   push edx
   push esi
   push edi
   MOV EAX,[AXON]
   mov edi,eax          {Basic address}
   mov esi,eax
   add esi,qPIECES      {Pieces}
   mov edx,white_black

@LP:                    {Loop start point}
   mov ebx,[eax]
   and ebx,edx
   jnz @fig
   add eax,4            {Entry point}
   JMP @LP
 @fig:
   cmp ebx,edx
   jz @finish           {Finish position checking !}
   mov ecx,eax
   sub ecx,edi    {Offset of the start point data}
   shl ecx,12
   or ebx,ecx       {Enter coordinates}
   mov [esi],ebx  {Save piece address}
   add esi,4
   add eax,4
   JMP @LP
@finish:
   sub esi,qPIECES
   sub esi,edi
   shr esi,2
   mov [edi+qPiecesNUM],esi    {Number of pieces}
   pop edi
   pop esi
   pop edx
   pop ecx
   pop ebx
   pop eax
end;
```

This ultrafast procedure processes the entire matrix of chess pieces and decodes it into a more convenient form - a list of features and addresses that are later used in other procedures. Based on these machine routines, the classic procedures for minimax, alpha beta and null move trackers are implemented. These procedures are written in Pascal, which is very easy to combine with machine code.

VI. Empiric Data and Experiments

During the testing of the influence of hardware on the execution speed of the Axon II chess engine, we collected data from several hardware platforms and systematized them in Table 1. *Benchmark tests were just focused on measuring the raw engine speed of the chess engine, the number of processed positions per second (NPS factor).* The tests were performed in the basic position (Fig. 1) up to a depth of 10, and in the Mate in 7 position up to a depth of 13 (Fig. 4). In parallel, the speeds of the referent PC engine Stockfish 18 were shown. *The main goal of the empirical research was to determine that engine is correct - find solutions in standard test positions. Also we want to determine on which hardware platform program executes the fastest.*

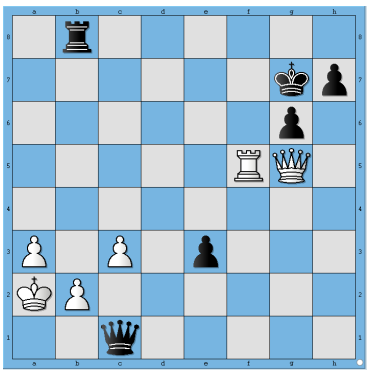

Fig. 4. Benchmark Mate-in-7 for Axon II engine.

The following list presents the Axon program logo file, which shows how the engine successfully solved this problem.

```
D=1  F5F2  (+0.11)          ==================
D=2  F5F2  (+0.05)          EVALUATION = +MT/13
D=3  F5F2  (+0.23)          SEARCH = +MT/13
D=4  F5F2  (+0.47)          TIME = 22.9 sec.
D=5  F5F2  (+0.74)          POS. = 76619392
D=6  F5F2  (+0.71)          NPS = 3.048.384 NPS
D=7  F5F2  (+0.81)          ==================
.....
D=12  F5F2  (+3.31)
D=13  G5F6  (+MT/13)
```

The results of benchmark testing on different hardware platforms are shown in Table I. It is clear that Axon II is a very fast engine, several times faster than the most modern Stockfish 18 engine. The speedup depends on many hardware parameters. The general conclusion is that the speedup is the highest on older Intel platforms (i3, about 5x). On the most modern AMD CPU, Stockfish 18 makes excellent use of the AVX512 instruction set for which its 64 code is optimized, but even here Axon II is about 3.5 times faster. Also, Axon II runs fastest on the most modern AMD Ryzen 7900X machine, where when solving the mate-in-7 problem with 12 pieces (Fig. 4) the speed exceeds **9 million positions per second**, on just one CPU core. Considering that this processor has 12 physical cores, there is huge potential for future parallelization.

TABLE I. BENCHMARK RESULTS.

Test Hardware (Single CPU Turbo Freq/MEM)	**Node per Second (NPS)**		
	Axon II basic position	***Axon II mate-in-7***	***Stockfish 18 basic position)***
Intel i7 2600 3.5 GHz 16 Gb DDR3 (1330 MHz)	1.238.488	3.157.504	358 kN
Intel i5 10500 4.5 GHz 32 Gb DDR4 (2666 MHz)	2.381.184	4.603.520	703 kN
Intel i5 11500 4.5 GHz 32 Gb DDR4 (2666 MHz)	2.970.453	6.076.672	854 kN
Intel i5 3470 3.4 GHz 8 Gb DDR3 (800 MHz)	**1.380.165**	**3.168.768**	**269 kN**
Intel i3 4150 3.5 GHz 16 Gb DDR3 (800 Mhz)	1.314.490	2.619.801	278 kN
AMD Ryzen 5 3600 4.2 GHz 16 Gb DDR4 (3192 MHz)	2.037.504	4.126.037	823 kN
AMD Ryzen 5 8500G 5 GHz 32 Gb DDR5 (4800MHz)	3.913.216	8.269.107	1030 kN
AMD Ryzen 9 7900X 5.5 GHz 32 Gb DDR5 (4800MHz)	**4.385.920**	**9.227.648**	**1260 kN**

VII. Conclusion

This paper presents the details of the construction of the novel Axon II chess engine, which is a continuation of research that was successfully evaluated in the Axon I engine series. After presenting the theoretical basis on which the construction of chess engines is based, the construction elements of the new author's engine, which is written in a combination of Pascal and X86 machine language, are presented, in order to achieve high performance. The validity and speed of the new engine are shown in basic benchmark tests using Stockfish 18 as referent engine. The first results are very promising and show extremely high speed values of the new Axon II engine (NPS factor) and therefore its potential for future upgrades. *Of course, Stockfish 18, which has dozens of heuristic techniques and special modifications of the basic algorithms, remains absolutely dominant in terms of total chess power in the world.* Future research direction will be focused on multicore and multithread parallelization of the Axon II engine.

Acknowledgement

This paper has been supported by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia [Grant Number: 451-03-34/2026-03/200102]

References

[1] P.W. Fray, An introduction to Computer Chess. Chess Skill in Man and Machine, Texts and monographs in computer science, Springer-Verlag, New York, N.Y, 1977.

[2] T. Marsland and J. Schaeffer, Computers, Chess and Cognition, Springer-Verlag, 1990.

[3] V. Vučković, "The Theoretical and Practical Application of the Advanced Chess Algorithms",*PhD Theses*,The Faculty of Electronic Engineering,The University of Nis,Serbia, 2006.

[4] V. Vučković, "The Compact Chessboard Representation",ICGA Journal,Volume31,Number 3,Tilburg,The Netherlands, ISSN 1389-6911. pp. 157- 164., 2008.

[5] R. Šolak and V. Vučković, "Time Management During a Chess Game",*ICGA Journal*,Volume 32,Number 4,Tilburg, The Netherlands, ISSN 1389-6911. pp. 206- 220., 2010.

[6] V. Vučković, "Axon Chess Engine Evaluation Function" Proc. of 11th International Conference on Electrical, Electronic and Computing Engineering(IcETRAN),Nis,Serbia,2024. DOI:10.1109/IcETRAN62308.2024.10735435