****

**Game Implementation Using**

**FPGA**

**Mini Project Report**

Submitted in partial fulfillment of the requirements

for

Semester V in Bachelor of Technology (ECE)

|  |  |
| --- | --- |
| **Supervisor**  Dr. Manish Goswami | **Submitted by**  Patlola Praveen(IEC2011025)  M Santhosh Mohan(IEC2011029) |

**Indian Institute of Information Technology - Allahabad**

Allahabad – 211012

**Certificate**

This is to certify that the project report entitled “**Game Implementation Using FPGA**”, is being submitted in fulfillment of Mini Project for the academic session July 2013- December 2013. The declaration made by the candidates is true to the best of my knowledge.

Date:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Dr. Manish Goswami**

Faculty

IIIT Allahabad

Allahabad

**Declaration**

We declare that this written submission represents our ideas and reference has been quoted where others' ideas or words have been included. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| **M Santhosh Mohan**  **IEC2011029** | **Patlola Praveen Reddy**  **IEC2011025** |

**Acknowledgements**

We would like to express our gratitude to our project mentor Dr. Manish Goswami. We are privileged to experience a sustained enthusiasm and involved interest from his side. This fueled our interest even further and encouraged us to boldly step into what was a totally dark and unexplored expanse before us.

We would like to thank our friends and elders for motivating us at every piece of work we picked up. Last but not least, we would like to thank the IIIT-A faculty members and the Institute, in general, for extending a helping hand at every juncture of need.

Table of Contents

[Chapter 1 7](#_Toc368862066)

[Introduction 7](#_Toc368862067)

[1.1 Objective 7](#_Toc368862068)

[1.2 Motivation 7](#_Toc368862069)

[1.3  Scope **Error! Bookmark not defined.**](#_Toc368862070)

[Literature Survey 8](#_Toc368862071)

[2.1 Iterative alpha beta pruning 8](#_Toc368862072)

[2.2 Massively parallel algorithms 8](#_Toc368862073)

[2.3 Existing chess engines 8](#_Toc368862074)

[2.3.1 Stockfish 8](#_Toc368862075)

[2.3.2 Crafty 9](#_Toc368862076)

[2.3.3 Zeta OpenCL chess 9](#_Toc368862077)

[2.4 Late Move Reductions 9](#_Toc368862078)

[Requirement Specification 10](#_Toc368862079)

[3.1 Hardware requirements 10](#_Toc368862080)

[3.2 Software requirements 10](#_Toc368862081)

[3.2.1 Scid package 10](#_Toc368862082)

[With Scid one can maintain a database of chess games, search games by many criteria, view graphical trends, and produce printable reports on players and openings. One can also analyze games with the Xboard or UCI compatible chess program, and even use Scid to study endings with endgame tablebases. 10](#_Toc368862083)

[3.3 Target OS 11](#_Toc368862084)

[Approach 12](#_Toc368862085)

[4.1 Overview of previous work 12](#_Toc368862086)

[4.1.1 Bitboard representation 12](#_Toc368862087)

**Abstract**

Field Programmable Gate­Array (FPGA) technology is gaining popularity among Application­Specific Integrated Circuit (ASIC) designers. Ease of development and maintenance makes FPGAs an attractive solution to many speed and efficiency­critical applications. The purpose of this project is to explore the world of FPGAs by implementing an arcade game on top of a VGA driver. The project was implemented on Xilinx Spartan­3E development board using Verilog hardware description language.

The project was started by learning Verilog as well as familiarising with the Spartan­3E development board and Xilinx ISE WebPACK design software. A number of simple applications were developed in order to comfortably proceed on to investigation of working principles of a VGA driver. It turned out that the synchronisation logic was rather trivial and that the custom implementation would not add much value to the project.

Verilog implementation of the classic Pong game was the first major task of the project. In this Game player tries to hit the moving ball with the paddle which will be under his control. A number of basic features, such as acceleration of the ball, textual information display, have been implemented. All the graphic elements will be designed from scratch.

## 

## Chapter 1

# Introduction

This chapter introduces the most important concepts of relevance to the project. It clears the existence of a project of this type including the reasons for picking this topic.

## Objective

The aim of this project was to explore the capabilities of modern programmable logic devices while getting hands­on experience of FPGA development.

The above definition is rather abstract and can be hard to assess. For this purpose, a number of objectives were set. Below, is the amended list of objectives.

• Learn about FPGAs, development tools and Verilog, start programming.

• Produce a simple application , that uses a VGA driver to display graphics on a computer monitor.

• Create a game .

Milestones were created to give a rough estimate of duration of the project.

## 1.2 Motivation

Good motivation for learning is a powerful part of any successful curriculum. Especially in project courses where we design and build an artifact, the time spent working on the project, and the depth of knowledge that comes out of the project, are dramatically increased if there is some strong internal motivation to go beyond the basics and delve into the project details. In particular we focus on a course where we learn about computer design by designing and building a computer from scratch on a Field Programmable Gate Array (FPGA).

Best way to learn about computer design is to design a computer. For us the opportunity to develop interactive games on a computer they designed is a strong motivator.

Easy way to get familiarize with the FPGA kit, Verilog coding , Xilinx WEBPack design software and finally with the I/O devices.

**Chapter 2**

# Literature Survey

This chapter aims to bring forth some of the existing work in the area of chess programming, studying them in more detail and in proper context and then drawing conclusions about existing and required features.

**2.1 Field-Programmable Gate-array(FPGA) Architecture – An Overview**

FPGA - is an acronym for Field Programmable Gate Array. It belongs to a class of user programmable digital devices called Programmable Logic Devices (PLD‘s) . A programmable logic device is an integrated circuit that enables the user to conﬁgure it in many ways,enabling the implementation of various digital logic functions, of varying sizes and complexities. PLD‘s can be classiﬁed into various categories :

1. Simple programmable logic devices (SPLD)

(a) Programmable logic array (PLA) : A programmable logic array is an integrated circuit that consists two levels of programmable logic ; an AND plane and an OR plane.

(b) Programmable array logic (PAL): A PAL is an integrated circuit that contains a ﬁxed OR plane followed by a programmable AND plane.

2. Complex Programmable Logic Device (CPLD)

3. Field Programmable Gate Array (FPGA)

**2.2 Field-Programmable Gate-array (FPGA)**

FPGAs are modern programmable logic devices that can be configured to perform any logic operation. An FPGA typically contains a matrix of programmable elements, also known as, Configurable Logic Blocks (CLBs). CLBs contain Look­Up Tables (LUTs), that can be used as logic or storage elements. The configuration data is stored in the memory.

Spartan­3E series FPGAs, used in this project contain the following structures :

* **Configurable Logic Block** – logic and basic storage elements are implemented using the Look­UP Tables (LUTs).
* **Input/Output Block (IOB)** – control the data flow between the I/O pins and internal logic of the device. LVTTL and LVCMOS logic standards are supported among others.
* **Block RAM** – memory used for data storage. Organized as 216kb dual ­ port blocks.
* **Digital clock manager block** – provides management for clock signals.

## 

## 2.3 Hardware Description Language :

Verilog, standardized as IEEE 1364, is a hardware description language (HDL) used to model electronic systems. It is most commonly used in the design and verification of digital circuits at the register-transfer level of abstraction. It is also used in the verification of analog circuits and mixed-signal circuits.

Verilog is case-sensitive and has a basic preprocessor (though less sophisticated than that of ANSI C/C++). Its control flow keywords (if/else, for, while, case, etc.) are equivalent, and its operator precedence is compatible. Syntactic differences include variable declaration (Verilog requires bit-widths on net/reg types [clarification needed]), demarcation of procedural blocks (begin/end instead of curly braces {}), and many other minor differences.

**Definition of syntax:**

The definition of constants in Verilog supports the addition of a width parameter. The basic syntax is:

<Width in bits>'<base letter><number>

**Digital systems theory**

Verilog is a good example of practical application of digital circuit theory. Many of the basic concepts, such as gate and register­transfer level design, combinational logic and finite state machines have been used throughout the project.

**Other HDLs :**

As with the FPGAs, there are two predominant hardware description languages – VHDL and Verilog **.**

**VHDL**

“VHDL is a language for describing digital electronic systems. It arose out of the United States government's Very High Speed Integrated Circuit (VHSIC) program. In the course of this program, it became clear that there was a need for a standard language for describing the structure and function of Integrated Circuits (ICs).

VHDL supports many built­in and user­defined data types. Some of most often used are

• std\_logic (single bit)

• std\_logic\_vector (bit vector)

• numerical types such as integer

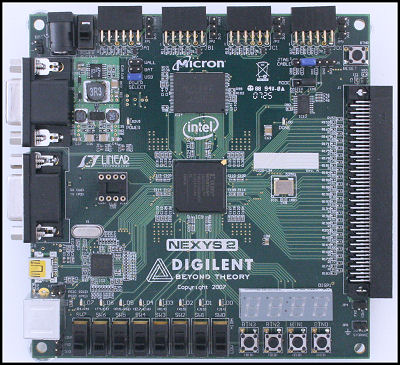
• arrays, enumerated lists, etc.

## 2.3 Development board

The Nexys-2 is a powerful digital system design platform built around a Xilinx Spartan 3E FPGA. With 16Mbytes of fast SDRAM and 16Mbytes of Flash ROM, the Nexys-2 is ideally suited to embedded processors like Xilinx's 32-bit RISC Microblaze. The on-board high-speed USB2 port, together with a collection of I/O devices, data ports, and expansion connectors, allow a wide range of designs to be completed without the need for any additional components.

**Specifications** :-

* Xilinx Spartan-3E FPGA 1200K gate
* USB2 port providing board power, device configuration, and high-speed data transfers
* Works with ISE/Webpack and EDK
* 16MB fast Micron PSDRAM
* 16MB Intel StrataFlash Flash R
* Xilinx Platform Flash ROM
* High-efficiency switching power supplies (good for battery-powered applications
* 50MHz oscillator, plus a socket for a second oscillator
* 75 FPGA I/O’s routed to expansion connectors (one high-speed Hirose FX2 connector with 43 signals and four 2x6 Pmod connectors)
* All I/O signals are ESD and short-circuit protected, ensuring a long operating life in any environment.
* On-board I/O includes eight LEDs, four-digit seven-segment display, four pushbuttons, eight slide switches
* Ships in a DVD case with a high-speed USB2 cable



Spartan 3E FPGA kit

### 2.42.4 VGA

### Hardware

### VGA is an analogue video standard, that is mostly used in personal computers. VGA can also refer to a piece of display hardware developed by IBM (Video Graphics Array) or a display mode, that uses 640 x 480 pixels resolution. VGA connector uses a total of 15 pins, but only 5 signals are needed for operation:

### • HSYNC – horizontal synchronization signal. This signal controls the horizontal position of the active pixel

### • VSYNC – vertical synchronization signal. This signal controls vertical position of the active pixel. VSYNC rate can also be referred to as a refresh rate (i.e. number of times per second the screen is redrawn)

### • RED – red colour channel

### • GREEN – green colour channel

### • BLUE – blue colour channel

### Other pins include ground, return paths and I2C clock/data or are reserved [5].

### HSYNC and VSYNC are TTL signals, so logic one is represented by 5V and logic zero is represented by 0V [5].

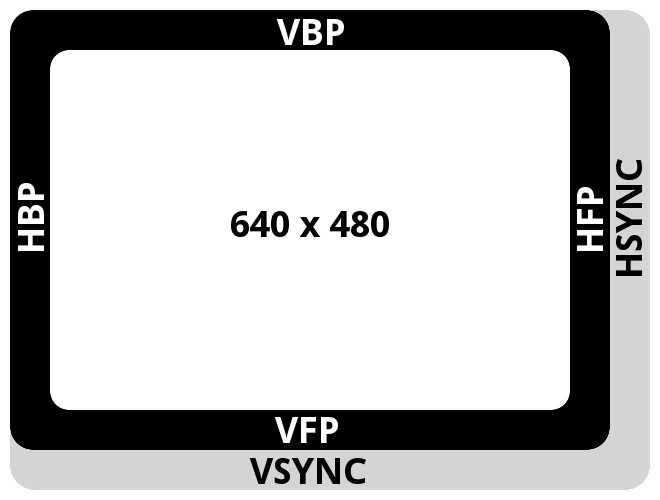
### RED, GREEN and BLUE signals are analogue. The maximum voltage that can be used is 0.7V and will result in full intensity of that colour .

Xilinx Spartan­3E board is only capable of producing eight colours (3­bits) as a digital­to analog converter (DAC) is not used. The board uses 270Ω resistors, which form a potential divider with internal 75Ω termination. This divider scales the 3.3V signal from FPGA to required 0.7V Back Porch) in Figure 2.2.**.**

**FIGURE**

**:**

**PARTS OF A SCREEN**



Timing

Pixels on the screen are drawn in sequence, one by one. Rows are arranged top to bottom, and columns go from left to right. The row and column addresses are constantly incremented thus changing the position of currently drawn pixel. Synchronisation signals are used to tell the monitor to return the pixel back to the first row (VSYNC) or the first column (HSYNC).

Sequence and duration of these signals are discussed below.

Visible part of the screen is shown as the white area. It has a resolution of 640 by 480 pixels.

The black and grey borders denote parts of the screen that are not visible, but required for synchronization. With these parts, the total width of the screen is 800 pixels, and the total height is 524 pixels. Below is the short description of each part of the screen, followed by a table of dimensions.

Active video. This is the visible part of the screen, video output is enabled.

Front porch. When the trace reaches the end of the visible part of the screen, the video output is disabled. These areas are denoted as VPF (Vertical Front Porch) and HPF (Horizontal Front Porch) in Figure.

Sync pulse. In case of HSYNC, the trace goes back to column zero. If pulse is VSYNC, the trace goes back to row zero. This part is also known as the retrace period.

Back porch. This is the part that goes before the active video starts. These areas are denoted as VBF (Vertical Back Porch) and HBF (Horizontal Back Porch) in Figure.

**Chapter 3**

# Requirement Specification

This chapter specifies the basic requirements for running the engine. The requirements are not specific - other alternatives may be used for the same facility.

### Hardware requirements

**3.1.1 OpenCL host CPU**

The central processing unit [[1]](http://en.wikipedia.org/wiki/Central_processing_unit#cite_note-espin2-0) (CPU) is the hardware within a [computer](http://en.wikipedia.org/wiki/Computer) system which carries out the [instructions](http://en.wikipedia.org/wiki/Instruction_(computer_science)) of a [computer program](http://en.wikipedia.org/wiki/Computer_program) by performing the basic arithmetical, logical, and [input-output](http://en.wikipedia.org/wiki/Input/output) operations of the system.

**3.1.2 OpenCL device GPU**

A graphics processing unit (GPU), also occasionally called visual processing unit (VPU), is a specialized [electronic circuit](http://en.wikipedia.org/wiki/Electronic_circuit) designed to rapidly manipulate and alter memory to accelerate the building of images in a [frame buffer](http://en.wikipedia.org/wiki/Frame_buffer) intended for output to a display. Modern GPUs are very efficient at manipulating [computer graphics](http://en.wikipedia.org/wiki/Computer_graphics), and their highly parallel structure makes them more effective than general-purpose [CPUs](http://en.wikipedia.org/wiki/Central_processing_unit) for [algorithms](http://en.wikipedia.org/wiki/Algorithm) where processing of large blocks of data is done in parallel.

### Software requirements

### 

### 3.2.1 Scid package

### With Scid one can maintain a database of chess games, search games by many criteria, view graphical trends, and produce printable reports on players and openings. One can also analyze games with the Xboard or UCI compatible chess program, and even use Scid to study endings with endgame tablebases.

**3.2.2 OpenCL SDK**

A software development kit is typically a set of [software development](http://en.wikipedia.org/wiki/Software_development) tools that allows for the creation of [applications](http://en.wikipedia.org/wiki/Application_software) for a certain [software](http://en.wikipedia.org/wiki/Software) package, software framework, hardware platform, [computer system](http://en.wikipedia.org/wiki/Computer_system), [video game console](http://en.wikipedia.org/wiki/Video_game_console), [operating system](http://en.wikipedia.org/wiki/Operating_system), or similar platform.

**3.2.3 C++11 compiler**

C++11[[1]](http://en.wikipedia.org/wiki/C%2B%2B11#cite_note-0) is the most recent iteration of the [C++ programming language](http://en.wikipedia.org/wiki/C%2B%2B_programming_language). The modifications and enhancements for C++ involve both the core language and the standard library.

**3.2.4 Polyglot 1.4**

PolyGlot is a "UCI adapter". It connects a UCI chess engine to an xboard interface such as WinBoard. UCI2WB is another such adapter(for Windows).

### 3.3 Target OS

The target operating system is Centos 6.x.CentOS is an Enterprise-class Linux Distribution derived from sources freely provided to the public by RHEL. The source codes however shall be portable to any operating system (including Microsoft Windws) provided that C++11 compatible compiler along with OpenCL SDKs are available. This is so as only the standard language extensions shall be used.

**Chapter 4**

# Approach

This chapter describes the modifications made to our earlier approach to a parallel chess engine. It briefly summarizes the previously accomplished work and the current approach used to embark upon the problem by enhancing the previous approach or switching on to entirely new efficient methods.

4.1 Overview of previous work

### 4.1.1 Bitboard representation

Bitboards[[[1]](#endnote-1)], in essence, are finite sets of up to 64 elements - all the squares of a chessboard, with one bit per a square. The bitwise operations are used to operate on bitboards. Bitwise boolean operators perform the intrinsic setwise operations, such as intersection, union and complement. Shifting bitboards simulates piece movement. We had earlier used 14 bitboards to represent a chess position, which has been now done with 4 bitboards only.

### Evaluation

Evaluation[[[2]](#endnote-2)] function is used to heuristically determine the relative value of a position, *i.e.*, the chances of winning. The white pieces have positive score whereas the black pieces have a negative score for the same. The Chess Board score is represented by a signed integer. The evaluation can be based on a couple of factors such as piece type, mobility, center control and king safety. The white pieces have positive score whereas the black pieces have a negative score for the same. The Chess Board score is represented by a signed integer.

### Piece-wise: Each piece is assigned a score such as:

|  |  |
| --- | --- |
| **Piece** | **Score** |
| Pawn | 100 |
| Knight | 325 |
| Bishop | 325 |
| Rook | 500 |
| Queen | 1050 |
| King | 40000 |

**Location:** The pieces are evaluated based on position via lookup evaluation table made for each piece and a separate table for center piece evaluation.

**Pawn structure**: This is used to describe the positions of all the pawns on the board, ignoring all other pieces. A pawn is doubled (or, sad to say, tripled) if there are more pawns of the same color on a given file. They usually introduce pawn structure weaknesses: some kind of backwardness, lack of a candidate passer that otherwise would be there hence get a penalty.

**Attacking value penalty**: A penalty in points is given for being attacked by any opponent piece.

**Defending value bonus:** Bonus points are given for being protected by one’s own pieces.

**Hanging piece penalty:** A hanging piece is an attacked piece that is not defended by own man exposed to capture.

score = c1\*material + c2\*mob + c3\*attk\_val + c4\*cent\_ctrl + c5\*pwn\_struct

### 4.1.3 Graphical user interface (GUI)

A chess engine is an application which is capable of playing chess and uses a standard protocol to communicate with an external Graphical User Interface (GUI). We use Scid as a GUI. With Scid one can maintain a database of chess games, search games by many criteria, view graphical trends, and produce printable reports on players and openings.

### 4.1.4 Implementing universal chess interface (UCI)

This is the description of a new interface between a chess engine and a graphical user interface called UCI. The benefits of using UCI[[[3]](#endnote-3)] interfacing include that all engine options can be modified within the graphical user interface,the GUI always knows exactly what the engine is doing , support for endgame table bases, independent of operating system, capable of network play etc. We use FEN notation for sending moves to the engine. It is a standard for describing chess positions using the ASCII character set. Here´s the FEN for the starting position:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

A FEN record contains 6 fields. The separator between fields is a space. The fields are:

* **Piece placement** (from whites’ perspective): Position of the pieces on the board.
* **Active color**: "w" means white moves next, "b" means black.
* **Castling availability**: If neither side can castle, this is "-". Otherwise, this has one or more letters: "K", "Q", "k", and/or "q".
* **En passant target square:** If there is no *en passant* target square, this is "-".
* **Halfmove clock:** This is the number of halfmoves since the last pawn advance/capture.
* **Fullmove number**: The number of the full move.

4.1.5 Opening books .bin format

Chess opening book[[[4]](#endnote-4)] refers to a database of chess openings used by chess programs. This eliminates the need for the program to calculate the best lines during approximately the first ten moves of the game, where the positions are extremely open-ended and thus computationally expensive to evaluate. We use a utility PolyGlot-1.4[[[5]](#endnote-5)] by Fabien Letouzey[[[6]](#endnote-6)], France which is a proxy chess engine that translates moves from Universal Chess Protocol to Scid/Xboard/Winboard Protocol.

To use opening book database in our chess engine we have to include it via UCI option:

option name OwnBook type check default\_path\_to\_binary\_book

### 4.1.6 Designing iterative version of alpha beta pruning

Alpha-beta pruning [1] is a search algorithm that seeks to decrease the number of nodes evaluated by the minimax algorithm in its search tree. As OpenCL kernel cannot execute recursion, the utmost need of the time was to define an alpha-beta pruning algorithm in iterative fashion. Donald E. Knuth and Ronald W. Moore explained[1] the steps involved in alpha-beta pruning and a method to implement it.

### 4.1.7 Search heuristic – killer move

Killer move heuristic[[[7]](#endnote-7)][[[8]](#endnote-8)] is an efficient way of ensuring we may not need to spend a lot of time searching lots of other moves first. The moves which repeatedly perform well at a certain *ply* are searched before the rest. This works as generally a good move in one position sometimes is equally a good move in other similar positions. It is hence a practically efficient idea to keep track of the moves which repeatedly cause beta cutoffs at each level, and place them nearer the top of the move list, depending on how often they are played.

### 4.1.8 Using OpenCL to parallelize tasks on CPU

Open Computing Language (OpenCL)[[[9]](#endnote-9)] was used as a framework to parallelize tasks across 12 cores of Intel i7 CPU. In order to spread the arena of computation now the focus is shifting to use of GPU exhaustively to extract performance.

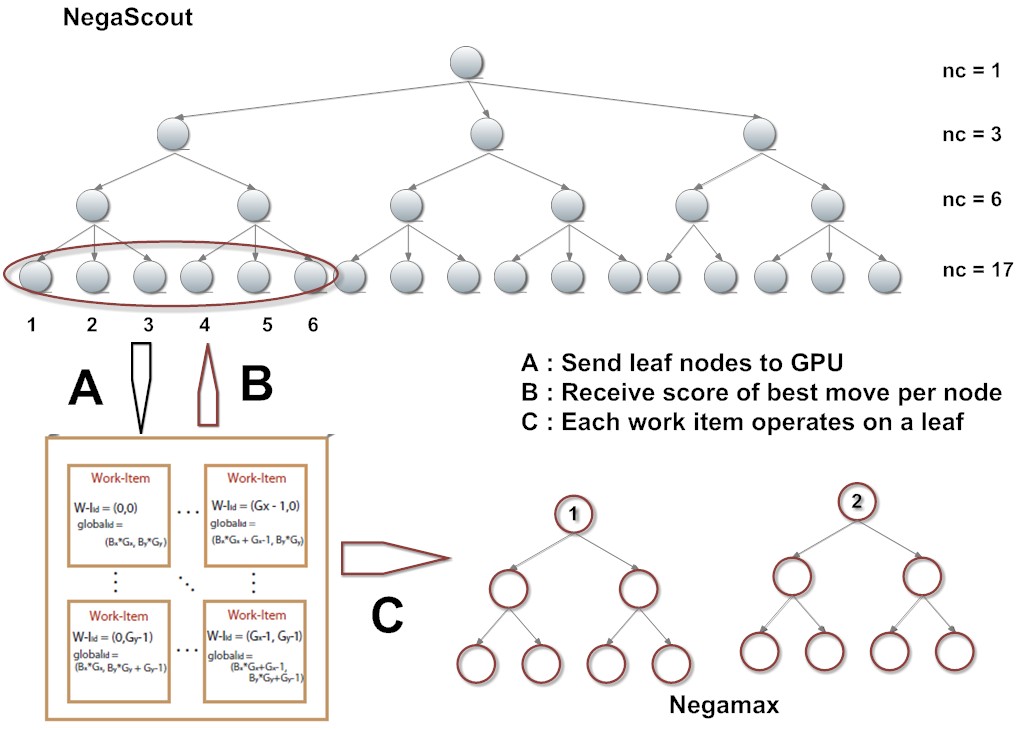
## Current add-ons

### 4.2.1 Tree splitting

Tree splitting[[[10]](#endnote-10)] is implemented in the sense that the search tree is created partially in CPU and partially in GPU. The CPU tree is constructed using C++11 threads by parallel generation of moves for each node. The CPU tree is created to a fixed CPU\_DEPTH and then the control is passed on to GPU in order to go deeper till GPU\_DEPTH to evaluate using the leaves from CPU as root nodes. The advantage of passing the control to GPU is that it can evaluate multiple nodes simultaneously and then switch back to CPU with evaluated scores. After the control is passed back to CPU the CPU sub-tree from the extreme leaf nodes to LCA (Lowest Common Ancestor) of the extremes is evaluated using NegaScout[**Error! Bookmark not defined.**] Algorithm. Once the score is assigned to the LCA of nodes the further computations on CPU tree are handled by NegaScout search. In order to determine which sub-tree’s leaves can be passed to GPU we take into account the LEVEL\_POPULATION. We can find the number of average number of child nodes per node by formula

avg\_children\_at\_level\_n = level\_population[n+1] / level\_population[n]

This would give us a *guesstimate* of the number of child nodes. We keep descending down in a sub-tree till we have a sub-tree whose number of leaf nodes is less than or equal to the number of work-items in one dimension of a work group.



**Figure 1.** CPU + GPU tree generation and evaluation

### Thread-safe transposition tables

We use C++11[[[11]](#endnote-11)] threads in CPU to parallel moves from a given position. Performing write operation to a global memory location by multiple threads running parallel comes with an overhead of using locks/mutex which again leaves us in an execution model of serial type. Thereby in order to eradicate the use of locks we would maintain a partially split transposition table[[[12]](#endnote-12)] in which we would have each thread at depth greater than a threshold would write to its fixed memory chunk. At lower levels as we would have comparatively less threads we would use a single transposition table with locking mechanism.

### Pre-order and post-order move-ordering

Pre-order move ordering[[[13]](#endnote-13)] will be implemented in both CPU and GPU virtual tree. Preorder move-ordering refers to ordering the moves generated per node before actually descending down to evaluate the child nodes. This would ensure evaluating the better nodes first and hence would help us get a cut-off at an earlier point of time. Pre-order can be done by using intelligent bubble sort on basis of evaluation scores or it can be a pre-order with an ordering factor of one meaning that the best node is swapped with the first node.

Post-order move ordering to be implemented in CPU tree where in order to generate the tree for the second time we simply garbage collect the garbage nodes from the earlier tree and grow the tree by one more level. If we order the nodes of the earlier tree we would get better cut-offs in the second traversal. This is called post-order move ordering as we would order the moves after the first traversal.

### 4.2.4 Search algorithms

We would use NegaScout algorithm for tree evaluation in CPU and NegaMax for tree traversal in GPU.The GPU tree would be heavy and running an search algorithm[[[14]](#endnote-14)] like NegaScout would complicate calculations. This is because in NegaScout each child node is evaluated twice.

NegaScout just searches the first move with an open window, and then every move after that with a zero window, whether alpha was already improved or not.

**Negamax**is a way of implementing [minimax](http://chessprogramming.wikispaces.com/Minimax) and derived algorithms. Instead of using separate subroutines for the Min player and the Max player it passes on the negated score due to the following mathematical relation:

max(a, b) == -min(-a, -b)

### Checkmate

Checkmate (frequently shortened to *mate*) is a situation in [chess](http://en.wikipedia.org/wiki/Chess) in which one player's [king](http://en.wikipedia.org/wiki/King_(chess)) is threatened with [capture](http://en.wikipedia.org/wiki/Capture_(chess)) (in [*check*](http://en.wikipedia.org/wiki/Check_(chess))) and there is no way to meet that threat. Or, simply put, the king is under direct attack and cannot avoid being captured. Delivering checkmate is the ultimate goal in chess: a player who is checkmated loses the game.

### Implementing late move reductions/history heuristics

A well-crafted alpha beta searcher in chess programs always has a mechanism to obtain a good order of all legal moves of a position. LMR makes use of this well-ordered move list for further pruning of sub-trees. The basic idea of this technique is based on the observation that a late move in the list hardly lifts the alpha value of the current node. Therefore, a sub-tree resulting from a late move might be a good candidate for pruning. Techniques which reduce the amount of work at fail low nodes. Until now, little work has been published in this area. Late move reductions are one of the few known techniques. The origin of this technique is lost in antiquity, and nobody seems to know who the inventor was. Late move reductions are based on the simple observation that in a program with reasonably good move ordering, a beta cutoff will usually occur either at the first node, or not at all. We make use of this observation by searching the first few moves at every node with full depth, and searching the remaining moves with reduced depth unless they look particularly forcing or interesting in some other way.

### ELO rating of engine

The Elo rating system is a method for calculating the relative skill levels of players in two-player games such as chess. It is named after its creator Arpad Elo, a Hungarian-born American physics professor. The Elo system was invented as an improved chess rating system. Each player has a numerical rating. A higher number indicates a better player, based on results against other rated players. The winner of a contest between two players gains a certain number of points in his or her rating and the losing player loses the same amount. The number of points won or lost in a contest depends on the difference in the ratings of the players, so a player will gain more points by beating a higher-rated player than by beating a lower-rated player. In chess, for instance, if one player is rated 100 points higher than the other player, they are expected to win about five games out of eight, and the rating changes reflect that. Over a series of games, if a player does better than expected, based on the ratings (compared to the opponents'), his or her rating will go up.

Chess rating classes:

|  |  |  |
| --- | --- | --- |
| **Elo rating** | **Class** | **Members** |
| 2200 - 2800 | Master | 4 % |
| 2000 - 2200 | Expert | 8 % |
| 1800 - 2000 | Class A | 12 % |
| 1600 - 1800 | Class B | 18 % |
| 1400 - 1600 | Class C | 18 % |
| 1200 - 1400 | Class D | 20 % |
| 0 - 1200 | Class E | 20 % |

[[15]](#footnote-1)Table of Chess ELO Rating

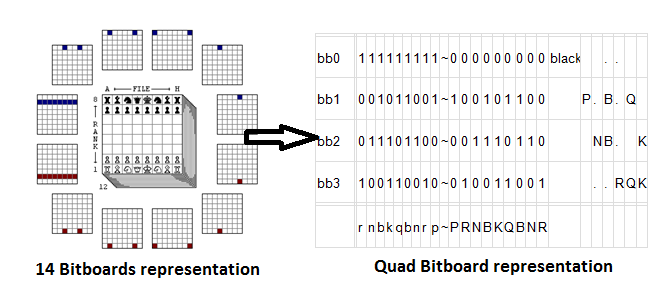
**Chapter 5**

# Description of Current Enhancements

This chapter describes the contribution to the project to date.

## 5.1 Porting 14 bitboard notation to quad bitboard

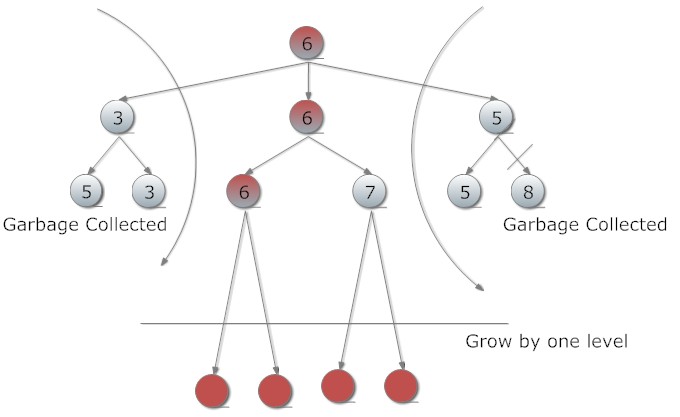
In the earlier notation for storing one chess position we used a combination of 14 64-bit bitboard which occupied 112 bytes. In order to use less space the whole code has been ported to a compressed form where we use only 4 64-bit bitboards to store a chess position. Now the space occupied is 32 bytes.

****

**Figure 2.** Bitboard representations

## 5.2 Host tree generation and garbage collection

Parallel move generation using C++11 standard threads is done in host. As already disclosed the host tree is created for the first search and then when the move is searched for the next time the previous tree is cleaned by the garbage collector and then grown by one more level in order to get the new host tree for the next search to be performed.

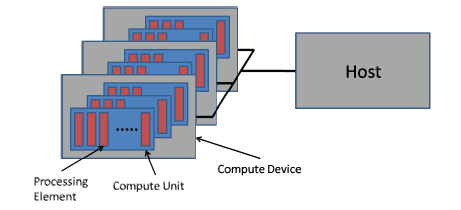


**Figure 3.** Garbage collection in host tree

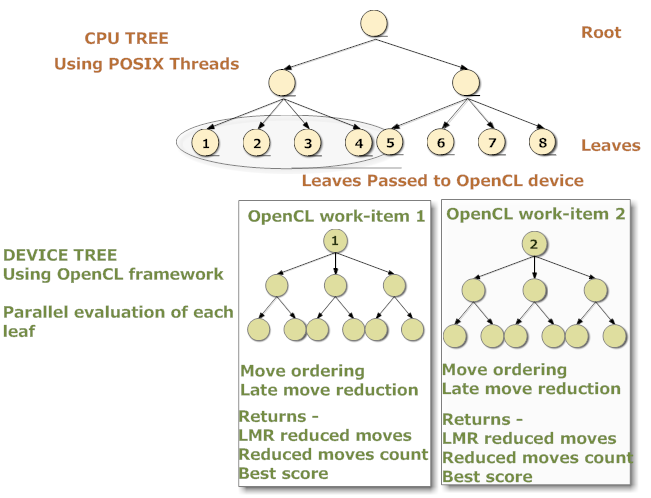
Once those branches which do not lead to the best move in the previous iteration are tracked they are automatically garbage collected by the garbage collector. And not the first child of the old tree is made the new node and the tree is sent to the function grow\_by\_one() in order to add one more level to tree so that the host tree reaches the CPU\_DEPTH cutoff.

## 5.3 Virtual tree in device

When an array of leaf nodes is sent to the device each leaf, an index in array is allocated to a work-item in the OpenCL device. Now each work-item treats the node it has with it as a root and expands the tree to the DEVICE\_DEPTH level. Each tree is evaluated at the work-item. Move ordering helps to get a cut-off in the initial level. We expand the device tree to a fixed depth, apply late move reductions to the moves and find the best potential moves to apply move ordering. Each sub-tree is expanded on each work-item and when the tree backtracking in device is over for each sub-tree a set of best late move reduced moves, with best scores are returned back. The array of scores holds the best score of the best move of the sub-tree.



**Figure 4.** Communication between device and host**[[16]](#footnote-2)**

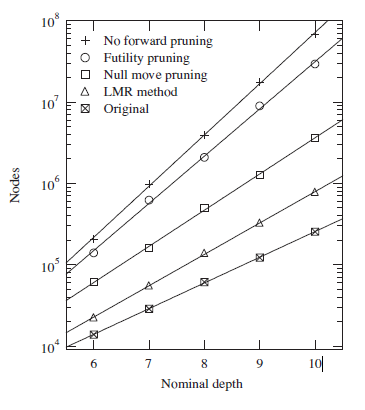


**Figure 5. Tree expansion in host and device**

### 5.4 Late move reductions

Late move reductions are based on the simple observation that in a program with reasonably good move ordering, a beta cutoff will usually occur either at the first node, or not at all. We make use of this observation by searching the first few moves at every node with full depth, and searching the remaining moves with reduced depth unless they look particularly forcing or interesting in some other way. If one of the reduced moves surprise us by returning a score above alpha, the move is re-searched with full depth.

Blindly reducing all moves after the first three or four will almost certainly have catastrophic results. We need some extra conditions for identifying tactically and positionally interesting moves, and avoid reducing these too often. Obviously, checks (and more generally, moves which are extended) should never be reduced. Most programs also avoid reducing captures and promotions.



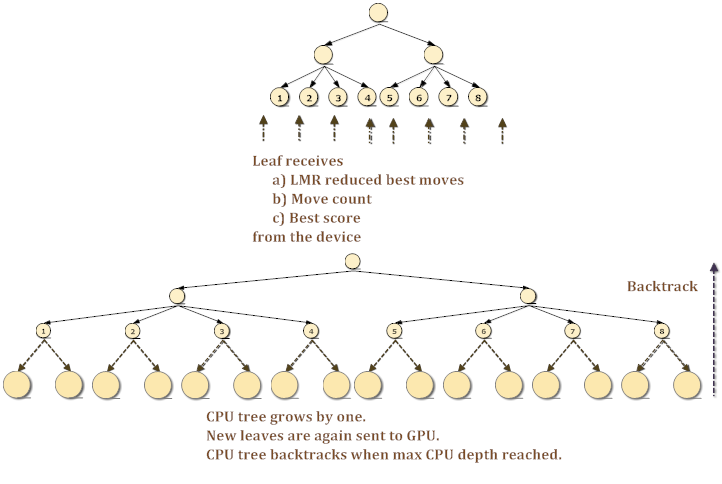
**Figure 6. The search–depth dependency of the number of nodes searched in chess.[[17]](#footnote-3)**

Beyond this, the conditions vary widely between different programs. Some of the more popular conditions are the following:

1. **History counters**- Moves which have failed high often in the past, or have a high (fail high) / (fail low) ratio are not reduced.
2. **Static or dynamic threat detection-** If the move which is a candidate for reduction contains some serious tactical or positional threat, reduction is avoided.
3. **Evaluation data-** This can be used in several ways. One possibility is to allow reductions only at nodes where the static eval is below alpha. It is also possible to check how each move affects the evaluation, and to avoid reducing those moves which improve the static eval or one of its components, from the point of view of the moving side.
4. **Node type-**It makes sense not to reduce any moves at PV nodes.

## 5.5 Host tree evaluation

After expanding host tree to a threshold CPU\_DEPTH depth, the leaves at lowest level are sent to device to be expanded parallely for late move reductions. Late move reductions are applied to get the potential nodes which should be expanded to full depth and thereby discarding the reducible nodes.



**Figure 7. Host Tree evaluation**

After late move reductions in device, the next level best nodes are returned to the CPU and then the tree grows by one more level. Similarly, the next generated level is again sent to the device to be reduced to best moves. The process stops when the maximum depth to be reached from CPU is reached. The the CPU tree backtracks to get the best move for the current root position.

### 5.6 Checkmate

In normal chess, the king is never actually captured – the game ends as soon as the king is checkmated because checkmate leaves the defensive player with [no legal moves](http://en.wikipedia.org/wiki/Rules_of_chess#Check). Deciding whether a check mate exists or not is a crucial step in evaluation. There may be a situation that there is a check mate but the opponent plays some other move. In order to consider all types of moves that can be made, after getting check mate we evaluate two levels deep and check out that if no leaf node exists with king in safe position then there is a check mate and the game is finished.

## 5.7 *en passant* and castling at backend

*en passant* is a special pawn capture which can occur immediately after a player moves a pawn two squares forward from its starting position, and an enemy pawn could have captured it had it moved only *one* square forward.

Castling consists of moving the king two squares towards a rook on the player's first rank, then moving the rook onto the square over which the king crossed. Castling can only be done if the king has never moved, the rook involved has never moved.

Earlier en-passant and castling were handled only by the user interface Scid. So if anybody wanted to play with the engine from the “console” he/she would not have en-pass and castling moves at his/her disposal. Besides this ignoring en-pass and castling while searching for the move is not a good practice as that may lead to ignoring certain good position while evaluation. Therefore we have hardcoded en-passant and castling to run at the engine level. Complete integration with testing is complete.

## 5.8 Dataset and profiling

For any software product, testing is the crucial part of the system. We have prepared a thoroughly tested test-database to test the code for running at various search depths. The best move at various depths exists in the test-base, so whenever a change is made to the code the code can be run on the existing test cases to detect the integrity and reliability of code.

## 5.9 ELO rating

Tools such as ELOstat and Bayeselo exists in market to find the ELO ratings of engine. They work on different methodologies to compare the games between individuals to give the ELO rating. We will use ELOStat to generate games and Bayeselo to find the ELO ratings.

**Chapter 6**

# Summary

On analysis of existing chess engines the expected features were jotted down. Stockfish, crafty, fruit, toga, Scidlet, phalanx are the string open source chess engines. All steps taken by us the this project aim at making a chess engine that would run on CPU and GPU and thus utilize the power of parallel processing to make complex calculations.

## 6.1 Comparative study

### 6.1.1 Our chess v/s Zeta chess engine

The table below highlights the differences between our chess engine and the standard OpenCL chess engine Zeta Chess which exists in the market. Currently Zeta is the only OpenCL chess engine in the market.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Our OpenCL engine** | **Zeta OpenCL engine** |
| Quad Bitboard | Yes | Yes |
| Opening Books | Yes | No |
| Heuristics | Yes | No |
| Transposition Table | Yes | No |
| Checkmate | Yes | No |
| Benchmarking | Yes | No |
| Horizon Effect (using QS) | Yes | No |
| Draw Moves | Yes | No |
| Garbage Collection | Yes | No |
| Castle and En-pass | Yes | To do |
| Original code? | Yes | Imports generation from Stockfish |

**Table 1.** Our OpenCL Chess v/s Zeta OpenCL Chess

### 6.1.2 Current work v/s previous work

The table below shows the difference between what is currently planned and done to the previous semester works. The throws light on the advancement of the work.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Version under development** | **Previous version** |
| Scalability | ~12x | 1 |
| Bitboard representation | 4 bitboard | 14 bitboard |
| Space per chess position | 32 bytes each | 112 bytes each |
| Space / game tree node | 16 bytes | 158 bytes |
| Sp. move Handling by engine | Yes | No |
| Garbage Collection | Yes | No |
| OpenCL device | CPU + GPU | CPU only |
| Heuristics | History heuristics | Killer move |
| Attack helper function | 2x faster and compressed\* | - |
| Benchmarking | Yes | No |
| Checkmate | Yes | No |
| Move ordering | Pre-order and Post-order | No |
| Transposition Table | Thread-safe | Simple |

Table 2. Current Work v/s Previous Work

## 6.2 Future Enhancements

### **6.2.1 Introducing chess closing databases**

An endgame tablebase is a computerized database that contains pre calculated exhaustive analysis of a chess endgame position. It is typically used by a computer chess engine during play, or by a human or computer that is retrospectively analysing a game that has already been played. The tablebase contains the game-theoretical value (win, loss, or draw) of each possible move in each possible position, and how many moves it would take to achieve that result with perfect play. Thus, the tablebase acts as an oracle, always providing the optimal moves.

### **6.2.2 Permanent brain/ Offline analysis/ Pondering**

In turn-based games, permanent brain is the act of thinking during the opponent's turn. Turn-based games such as chess have a weakness: one of the players can spend too much time thinking.

The strength of chess programs depends very much on the amount of time allocated for calculating. Many chess programs use pondering to improve their strength. Current programs cannot create strategic plans, so a program simply tries to predict the opponent's move and begins to calculate its response. If the opponent's move has been guessed correctly, then the program continues to calculate. If the prediction fails, the program begins a new computation.

## 6.3 Conclusion

The search algorithm is being modified completely and now more stable search algorithms such as negamax and NegaScout would be used. Besides this heuristics which make pruning fast are added to the search. If the product is going to completely abide by its specifications along with a remarkable performance it would be released in open source.

# Appendix

## Open Source Components

### Scid

We have used Shane's Chess Information Database (Scid) to display the moves in graphics. Besides this, Scid is a package with wide range of features such as maintaining database of chess games, search games by many criteria, view graphical trends, train and produce printable reports on players and openings. Scid will prove to be very helpful when we will take into consideration the end game databases.

### PolyGlot

PolyGlot is a "UCI adapter". It connects a UCI chess engine to an interface such as Scid. Besides we have used polyglot for making opening books, *i.e.* converting .pgn files to .bin format.

# References

1. [] Pradyumna Kannan, Magic Move-Bitboard Generation in Computer Chess. (*Casual paper writeen on April 30, 2007)* [↑](#endnote-ref-1)
2. [] Philip Paulsen, Johannes Fürnkranz. A Moderately Successful Attempt to Train Chess Evaluation Functions of Different Strengths, *ICML-10 Workshop on Machine Learning and Games.* [↑](#endnote-ref-2)
3. [] Stefan-Meyer Kahlen,UCI Protoco, April 2004. Available:

   http://wbec-ridderkerk.nl/html/UCIProtocol.html (Last seen on- 25th Nov. 2012) [↑](#endnote-ref-3)
4. [] Chess Opening Books. Available at: http://www.chess.com/openings/ [↑](#endnote-ref-4)
5. [] Fabien Letouzey. UCI Adapter, PolyGlot 1.4

   Available: http://wbec-ridderkerk.nl/html/details1/PolyGlot.html (Last seen on - 19th August 2012) [↑](#endnote-ref-5)
6. [] Polyglot Book Moves Format. Available: http://alpha.uhasselt.be/Research/Algebra/Toga/book\_format.html (Last seen on – 30th Oct 2012) [↑](#endnote-ref-6)
7. [] Jonathan Schaeffer. The History Heuristic and Alpha-Beta Search Enhancements in Practice. *IEEE Trans. Pattern Anal. Mach. Intell. 11(11): p1203-1212 (1989)* [↑](#endnote-ref-7)
8. [] S.G. Akl and M.M. Newborn. The Principle Continuation and the Killer Heuristic, *ACM*

   *Annual Conference*, 1977, 466-473. [↑](#endnote-ref-8)
9. [] OpenCL Best Practices Guide (Version 1.0). Available: http://www.nvidia.com/content/cudazone/CUDABrowser/downloads/papers/NVIDIA\_OpenCL\_BestPracticesGuide.pdf (Last seen on - 3rd Dec 2012) [↑](#endnote-ref-9)
10. [] R. M. Hyatt. The dynamic tree-splitting parallel search algorithm*. ICCA Journal, 20(1):3–19, 1997.20(1):3–19.* [↑](#endnote-ref-10)
11. [] Marc Gregoire, Nicholas A. Solter, Scott J. Klepe, “Multi-threaded programming with C++” in *Professional C++,2nd Ed,*  p797. [↑](#endnote-ref-11)
12. [] R. M. Hyatt and T. Mann. A lockless transposition-table implementation for parallel search*. ICGA Journal*. 23(2):67–74,Vol. 25, No. 1 - March 2002 [↑](#endnote-ref-12)
13. [] Eric Thé. “An Analysis of Move Ordering on the Efficiency of Alpha-Beta Search,*”* MS Electronic Thesis. [↑](#endnote-ref-13)
14. [] Brian Greskamp. Parallelizing a Simple Chess Program. Available email: greskamp@crhc.uiuc.edu

    **Comments** [↑](#endnote-ref-14)
15. Table taken from US Chess Federation database at http://senseis.xmp.net/?EloRating [↑](#footnote-ref-1)
16. The image is taken from OpenCL Best Practices Guide (v1.0) [↑](#footnote-ref-2)
17. Crafty is used as a base program of this experiment. J. Entertainment Computing 3, p55 [↑](#footnote-ref-3)