# Accelerare hardware a explorării în spațiul stărilor pentru jocul Reversi

Marius M. TIVADAR
Universitatea „POLITEHNICA" Timișoara, România
Conducător științific: conf. dr. ing. Doru TODINCĂ
- Lucrare de diplomă -

5 iunie 2009

## Introducere

Ideea de „gândire artificială" în jocurile precum Șah, Go, Reversi a fascinat lumea încă din cele mai vechi timpuri în momentul în care nu existau calculatoare moderne, iar posibilitatea rezolvării unui joc de către o mașină era un mister total. În anul 1770, mașina *The Turk* era cunoscută ca și „mașina automată de șah", a fost probabil una din cele mai importante invenții ale omului care a răspândit întrebarea, „Oare este capabilă o mașină să gândească?". Bineînteles, *The Turk* era o farsă extraordinar de bine realizată, o mașinărie construită de către *Wolfgang von Kempelen* care părea capabilă de un joc de șah foarte competent, precum și de rezolvarea problemei *Turul Cavalerului*.[1] Cel puțin așa se credea inițial, pentru că mașinăria avea defapt în ea ascuns un om, care printr-un mecanism complicat, instruia operatorul ce să facă. Farsa a bucurat oamenii timp de 84 de ani, însuși Napolean Bonaparte jucând o partidă cu mașinăria. A fost un pas mare din punct de vedere filosofic, întrebarea a rămas, frământând mințile oamenilor mult timp.

În 1927, matematicianul și filosoful John von Neumann, a enunțat în lucrarea sa „Zur Theorie der Gesellschaftsspiele" teoria *minmax*, iar în 1944 împreună cu Oskar Morgenstern a fundamentat domeniul Teoria Jocurilor. Din acest moment exista posibilitatea algoritmizării unui joc de șah.[16][1]
Claude Shannon, a publicat în 1949 lucrarea intitulată „Programming a Computer for Playing Chess" în care a descris funcționalitatea unui program de șah pe un calculator, folosind teoriile lui Neumann. Shannon a observat imposibilitatea unui calculator de a explora întreg spațiul al stărilor și a propus metoda de explorare parțială, evaluând starea tablei de joc printr-o funcție euristică propusă. Programul, nefiind realizat, era considerat capabil să joace împotriva unui adversar începător.[5]
În 1997 IBM a proiectat calculatorul *Deep Blue*, un adversar comparabil cu campionul mondial *Garry Kasparov*. Mașinăria era un monstru, o arhitectură masiv paralelă conținând 30 de procesoare care funcționau la frecvența de $120MHz$, la care se adăugau 480 de chipuri VLSI specializate pentru jocul de șah. Rezultatul a fost $3.5 - 2.5$ pentru *Deep Blue*.[2]

[1] Problema acoperirii tablei de șah cu un cal, astăzi o problemă trivială rezolvabilă prin forță brută, existând și metode mai evoluate.

[2] 0.5 reprezintă remiză.

## Reversi

Reversi este un joc derivat din Go, inventat în Anglia în anul 1883. Regulile au fost modificate în timp și este cunoscut sub numele de Othello sau Reversi. Regulile sunt foarte simple: se joacă pe o tablă $8 \cdot 8$, starea inițială fiind formată din 4 piese puse în mijloc, două albe și două negre. Jucătorii se numesc *alb* și *negru*[3]. Fiecare jucător adaugă pe rând un disc de culoarea lui. Regula este ca discul pus să captureze cel puțin un disc adversar. Capturarea constă în transformarea discurilor adversare în culoarea jucătorului care a mutat și se întâmplă pe toate cele 8 direcții prin flancarea cu un alt disc de aceași culoare.[4] Jocul se termină atunci și numai atunci când nici un jucător nu mai are mutări valide. Câștigă jucătorul care are cele mai multe discuri pe tabla de joc. Regulie sunt simple, un joc bun este extrem de dificil. Din 1977 există campionatul mondial de Reversi.[2] Despre Reversi se spune:

> „A minute to learn... A lifetime to master."[5]
>
> Becker.

## Complexitatea

Reversi face parte din clasa jocurilor cu informație perfectă, cu doi jucători și cu sumă zero.[6] Reversi alături de Go și Șah sunt campionii, reprezentând jocurile care în prezent nu au rezolvare, spațiul stărilor fiind prea mare pentru a fi explorat în totalitate.[4] Matematic jocul este nerezolvat, nu se cunoaște rezultatul jocului în cazul în care fiecare jucător joacă perfect. Nu se cunoaște nici dacă o mutare din începutul jocului are o influența în sfârșitul jocului. Spațiul stărilor este de $10^{28}$ iar complexitatea arborelui de joc este de $10^{58}$.[1]

## Adversari artificiali în Reversi

Reversi este probabil cel mai *încercat* joc. Oamenii care îl studiază sunt atrași de regulile foarte simple ale acestuia și complexitatea foarte mare. Multe constante din acest joc sunt puteri ale lui 2: tabla are 64 de poziții, sunt 2 jucători, 2 tipuri de piese, ceea ce duce la o reprezentare ușoară a jocului în sistemele digitale. Toată concentrarea rămânând pe strategia de joc și viteză. Spre deosebire de Șah sau Go, computerele de Reversi depășec maeștrii. În 1997 Michael Buro a creat programul *Logistello* care a bătut campionul mondial Takeshi Murakami cu scorul de $6 - 0$. Programul rula pe o stație SPARC10 și era scris în C. În 1993 s-a înființat primul turnament de Reversi jucat de computer, intitulat „International Paderborn Computer-Othello Tournament" la Universitatea din Paderborn, Germania. Concursul a avut pasionați din diferite țări, fiecare cu idei inovatoare și cu diferite arhitecturi. Locul I a fost câștigat de *Logistello*, avându-l pe creator pe Michael Buro, fondatorul concursului. Numeroase cercetări au urmat în domeniul academic legate de această temă.[17][3][18]

## Tema Proiectului

Ce îmi propun eu în acest proiect, este de a proiecta un calculator capabil să joace Reversi cu o accelerare puternică hardware, în dorința de a fi mai rapid ca un PC. Funcționalitatea este implementată în FPGA, iar mașina este autonomă, fiind legată doar la un monitor pentru a vizualiza tabla. Pentru explorare în spațiul

[3] Sau roșu și albastru.

[4] Orizontal stânga dreapta, vertical stânga dreapta și pe diagonale în ambele sensuri.

[5] Un minut pentru a-l învăța, o viață pentru a-l stăpâni.

[6] Adică câștigul unui jucător este pierderea adversarului.

stărilor folosesc algoritmul clasic *minmax* cu optimizare *alpha-beta*[20] ce va reduce complexitatea algoritmului clasic. Voi proiecta practic un chip specializat, prototip implementat în FPGA, capabil să joace acest joc. Modulele din care este compus, sunt specializate și fiecare au funcții specifice jocului. De exemplu, modulul de generare a mutărilor valide, modulul de tranziționare de la o stare la alta, modulul de evaluare euristică a tablei de joc. Fiecare modul este gândit încât să exploateze avantajul hardware în fața software-ului pe un CPU general. Mașina are un semnal de ceas de $50MHz$. Voi identifica modulele considerate cele mai costisitoare computațional și am să încerc o paralelizare internă a acestora. Modulul de evaluare a stării de joc, conține o serie de parametrii determinați empiric, iar proiectarea lui va exploata puternic capabilitățile hardware. Totuși, pentru creșterea frecvenței va fi nevoie de și mai multă proiectare, dar nu este scopul lucrării de față. În final, voi compara cu același algoritm, aceleași euristici, dar implementate pe un PC. Rezultatul este pe departe în favoarea jocului implementat în FPGA.
La sfârșitul lucrării, voi încerca o abordare teoretică a explorării paralele a spațiului stărilor în FPGA. Problema rămânând deschisă din punctul de vedere al lucrării mele.[14]

## Arhitectura

Pentru o explorare accelerată, va trebui să determinăm modulele care sunt costisitoare computațional. Aceste module vor fi apoi utilizate de algoritmul principal de explorare, construit cu un automat finit.

## Harta mutărilor

Primul modul considerat, este cel care ne va returna mutările posibile de pe tabla de joc ale unui jucător. Din algoritmul de explorare, observăm că la fiecare nod din arborele de joc va trebui să calculăm mutările valide pentru starea viitoare. Pe un CPU general, vom scrie un program care va determina pentru fiecare poziție în parte dacă este o mutare validă sau nu. În acest procesor specializat pentru Reversi, voi determina acest lucru în paralel, determinând o expresie logică pentru fiecare poziție. Intrarea modulului va fi: $2 \cdot 64$ biți care reprezintă tabla de joc, 1 bit jucătorul. Ca și ieșire vom avea o hartă de 64 de biți care va avea valoarea 1 pentru fiecare poziție validă. Modulul va calcula o hartă la fiecare ciclu de ceas.
Tabla de joc este reprezentată prin $2 \cdot 64$ biți, câte 64 de biți pentru fiecare culoare. Această reprezentare minimă ne va permite să construim algoritmi care se implementează ușor și cu o latență mică.

## Modulul de tranziție între stări

Acest modul este cel mai utilizat în algoritm, pentru că el va face tranziția dintre o stare de joc în alta. Prin urmare, va fi utilizat intens, el generând fiecare nod din arborele de joc. Viteza algoritmului final va fi invers proporțională cu latența acestui modul, prin urmare dorim o proiectare foarte bună. Ideea a fost inspirată din sumatorul *ripple carry*, care propagă bitul de transport la următoarele celule[11]. Putem reprezenta tabla de joc ca o rețea de celule interconectate fiecare cu toți cei 8 vecini.[7] Considerăm un semnal care propagă *valul*, iar în momentul în care a ajuns la o celulă ce conține un disc de aceași culoare ca și jucătorul care e la rând, *valul* se va propaga invers. Toate celulele care au *prins* ambele valuri își vor complementa starea, adică discul își schimbă culoarea. Acest algoritm simplu și rapid este posibil datorită proprietăților jocului

[7] Avem 8 direcții în care se pot captura discuri.

și a reprezentării tablei de joc ca două semnale a câte 64 de biți. Pentru a simplifica circuistica, vor exista 8 straturi de celule, fiecare strat conectate doar într-o direcție. Pentru exemplificare, considerăm un singur lanț:

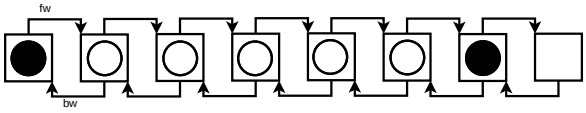


Figura 1: Considerăm propagarea valului în acest exemplu, de la stânga spre dreapta, până la penultima celulă unde se află un disc de culoarea considerată curentă, negru. *fw* marchează direcția de propagare, iar *bw* semnalul ce se întoarce în cazul în care a fost găsit un disc de culoare neagră.

Circuitul este combinațional, iar rezultatul este memorat în registre. La intrare, modulul primește configurația unei table de joc, adică $2 \cdot 64$ biți, semnal pe 1 bit reprezentând jucătorul, și *X, Y* ambele semnale pe 3 biți reprezentând coordonatele unde se dorește mutarea. La fiecare ciclu de ceas vom avea la ieșirea modulului o nouă stare de joc. Aceste două module prezentate, fiecare având un rezultat într-un cc[8], sunt abordări total diferite față de o descriere algoritmică pentru un CPU general. Ne vom folosi de acest avantaj a circuitelor specializate, pentru a crea un adversar Reversi puternic.

# I/O

Vizualizarea tablei de joc se face pe monitor, *controller-ul* de VGA fiind un alt modul descris în lucrare. Folosesc modul VGA standard, rezoluție 640x480@60Hz. Jocul este implementat astefel încât se poate juca cu un jucător uman. Comenzile pot fi introduse cu ajutorul a 4 butoane de pe placa de dezvoltare. Circuitul de *debounce* a fost construit simplist, cu un divizor de frecvența construit printr-un numărător. Se va citi starea semnalului cu o frecvență mică.

# Euristica

Modulul pentru evaluarea unei stări a jocului va fi folosit la fiecare nod terminal din arbore, adică foarte des. Il considerăm deci un modul critic, iar proiectarea lui va exploata ca și primele două module, capabilitățile designului hardware și reprezentarea foarte bună a tablei de joc. Euristica presupune compararea stării de joc cu o serie de șabloane studiate[3][6][17]. Se va da un scor mare pentru stabilitatea poziției, adică discurile care nu mai pot fi întoarse de adversar[9], scor care va fi penalizat cu scorul discurilor stabile ale adversarului. Un alt factor în această euristică, este mobilitatea[3][17], care presupune numărul de mutări valide în acea stare. Șabloanele, sunt reprezentări tot pe 64 de biți, putem astfel testa foarte ușor dacă un șablon e conținut în tabla de joc. Fiecare șablon va genera alt scor, iar în total sunt aproximativ 100 de șabloane. Avem nevoie deci, de a calcula operații *AND* între reprezentarea tablei de joc și aceste șabloane și apoi de a aduna scorurile. Aceste operații vor fi executate în paralel, mai multe sumatoare ce vor calcula scorul tablei de joc. În lucrare voi prezenta mai multe metode de proiectare încercare, precum și optimizări posibilie, de exemplu arbori *Wallace*[19][15][12][10]. Aceste calcule implementate pe PC, CPU general, necesită multe cicluri *for*, aici beneficiem de paralelism și vom obține un rezultat la fiecare ciclu de ceas. Trebuie să menționez că tema lucrării nu presupune construirea unui evaluator bun a stării de joc, dar cel prezentat funcționează foarte bine, câștigând chiar în fața mai multor programe de Reversi găsite pe internet.

[8] clock cycle.

[9] De exemplu colțurile și tot ce se construiește pe lângă ele.

## Explorarea

Viteza de calcul a modulelor prezentate până acum, vor duce la o explorare în spațiul stărilor foarte rapidă. A fost implementat un algoritm *minmax*, cu tăiere *alpha-beta*. Implementarea este formată dintr-un automat finit, fiecare stare controlând un modul din cele prezentate, sau memorând în $BRAM$[10] stări din arborele de joc.[9]

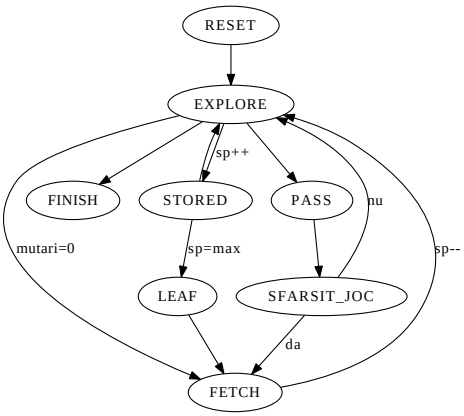


Figura 2: O reprezentare aproximativă a stărilor automatului de explorare.

Algoritmul pornește din starea *RESET* și va trece prin *EXPLORE*, care va lua o mutare posibila din cele determinate, folosind un decodificator prioritar și va tranziționa în altă stare de joc, nivelul arborelui fiind memorat. Eticheta *sp++* ilustrează o adâncire în arbore, iar *sp–* o revenire. În starea *LEAF* se va evalua poziția de joc, după care se va reveni în arbore. În realitate sunt mai multe stări, fiind nevoie de ciclii de ceas suplimentari pentru module. În momentul în care nu mai sunt mutări și s-a ajuns din nou la rădăcina arborelui, atunci explorarea se termină și ajungem în starea *FINISH*.

[10]Block RAM, memorie utilizabilă în interiorul FPGA-ului.

## Concluzii

Designul prezentat funcționează la o frecvență de $50MHz$. S-a rulat un algoritm obișnuit de *place&route* și nu s-a făcut nici o optimizare la nivel de FPGA. Comparația s-a făcut cu un program scris pe un PC cu procesor Intel Pentium 4 la o frecvență de 3GHz , programul fiind scris in limbajul C# platforma .NET. Implementarea hardware s-a făcut pe o placă de dezvoltare Spartan3E cu un FPGA xc3s500e și au fost folosite aproximativ 60% din resursele acesteia. Amândouă implementări au fost setate să exploreze cu o adâncime de 6 niveluri în arbore.

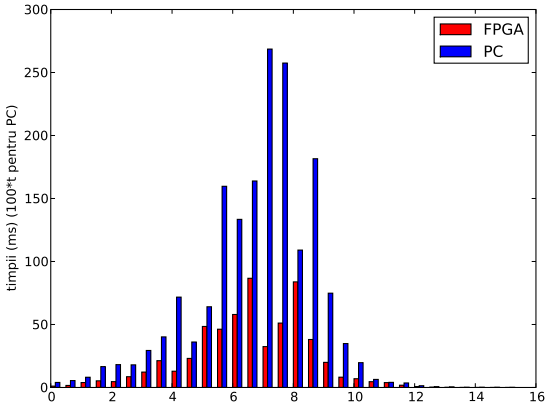


Figura 3: Timpii pentru fiecare mutare, comparație FPGA/PC.

În această figură, este prezentată o partidă de joc, fiecare bară reprezentând timpii de gândire pentru fiecare implementare. Pe axa $X$ se află numărul mutării. Și algoritmul de pe calculator și cel implementat în FPGA, au jucat aceleași mutări iar adversarul lor fiind eu. Timpii pentru PC sunt de 100 de ori mai mici în grafic, pentru a avea sens reprezentarea. Dacă cumulăm toți timpii de gândire pentru fiecare arhitectură, reiese că jocul implementat în FPGA este de aproximativ 250 de ori mai

rapid. Bineințeles, rezultatul are doar o valoare informativă. Trebuie menționat că pe PC timpii sunt mai mari și pentru că este vorba de o implementare nu tocmai optimă. Platforma software .NET oferă o flexibilitate foarte mare, dar în spate va face multe alte operații care îi sunt transparente utilizatorului. Pentru FPGA, timpii sunt determinați exact: se incrementează un numărător cât timp algoritmul explorează spațiul stărilor, după care valoarea este transmisă pe interfața serială la un calculator.

## Bibliografie


[1] L. V. Allis, *Searching for Solutions in Games and Artificial Intelligence.* PhD thesis, 1994.

[2] B. Rose, *Othello: A minute to learn... A lifetime to master.* 2005.

[3] R. Fang, *Othello: From Beginner to Master.* 2003.

[4] J. v. R. H. Jaap van den Herik, Jos W.H.M. Uiterwijk, "Games solved: Now and in the future," 2002.

[5] C. Shannon, "Programming a computer for playing chess," in *Philosophical Magazine.*

[6] M. Buro, "Experiments with multiprobecut and a new high-quality evaluation function for othello," tech. rep., NEC Research Institute.

[7] P. M. Nyasulu, *Introduction to Verilog.* 2001.

[8] K. S. Iyer, *High Performance FPGA Designs.* 2006.

[9] Xilinx, *XST User Guide.* 2008.

[10] K. Hwang, *Computer Arithmetic.*

[11] J.-P. Deschamps, *Synthesis of Arithmetic Circuits.* 2006.

[12] P. V. G. Oklobdzija, *High-Speed VLSI Arithmetic Units: Adders and Multipliers.* 1999.

[13] Motorola, *Verilog HDL Coding.* 1999.

[14] B. Parhami, *Introduction to Parallel Processing.* 2002.

[15] E. Hwang, *Principles of Digital Logic Design.* 2003.

[16] K. L.-B. Yoav Shoham, *MULTIAGENT SYSTEMS: Algorithmic, Game-Theoretic, and Logical Foundations.*

[17] M. Buro, "The evolution of strong othello programs," tech. rep., NEC Research Institute.

[18] M. Buro, "An effective selective extension of the $\alpha - \beta$ algorithm," tech. rep.

[19] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2001.

[20] C. A. Giumale, *Introducere în Analiza Algoritmilor.* POLIROM, 2004.