Implementing Propositional Networks on FPGA

Cezary Siwek[1], *Jakub Kowalski*[1],
Chiara F. Sironi[2], Mark H. M. Winands[2]

[1]University of Wrocław, Poland
[2]Maastricht University, The Netherlands

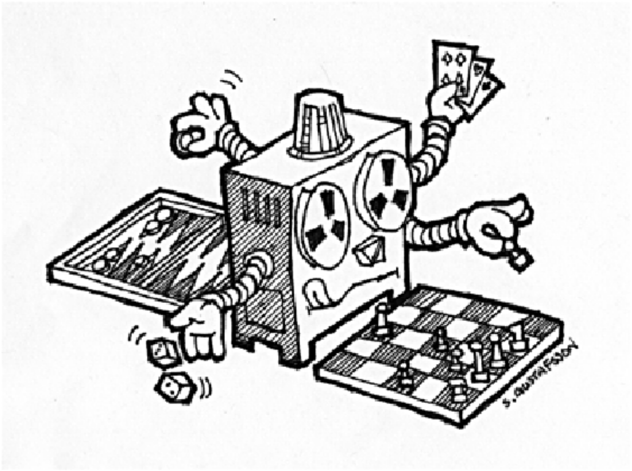
AUS-AI

14.12.2018

What is this work about?

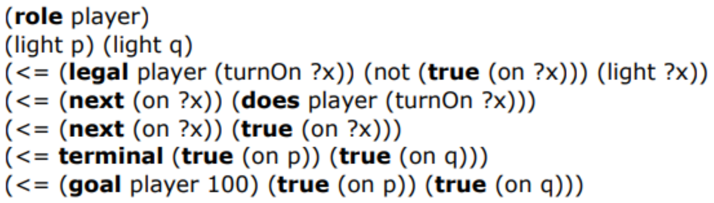
1. Take the formalism describing a class of games
2. And its reasoning engine represented as a logic circuit
3. Translate to, and implement on a hardware
4. Get efficiency profit

General Game Playing (GGP)



GDL (Genesereth, Love, Pell; 2005)

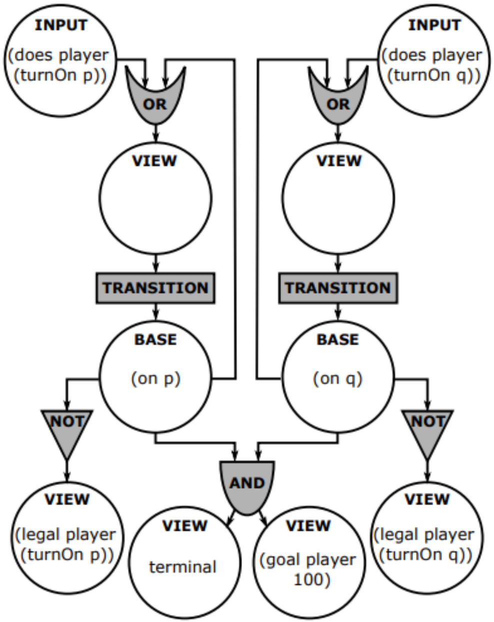
- Describes any turn-based, finite, and deterministic *n*-player game with perfect information.
- Datalog-based, high-level, strictly declarative language
- International General Game Playing Competition (2005-2016, 2019?)
- State of the game is a set of true facts
- No predefined concepts, only a few keywords
- Keywords define different game elements and the game dynamics



```
(role player)
(light p) (light q)
(<= (legal player (turnOn ?x)) (not (true (on ?x))) (light ?x))
(<= (next (on ?x)) (does player (turnOn ?x)))
(<= (next (on ?x)) (true (on ?x)))
(<= terminal (true (on p)) (true (on q)))
(<= (goal player 100) (true (on p)) (true (on q)))
```

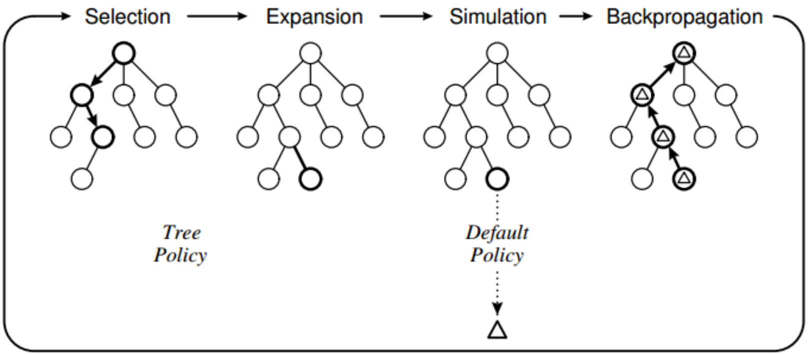
Propositional Nets (Schkufza, Love, Genesereth; 2008)

- Alternative representation for game dynamics
- Actually a logical circuit
- Directed graphs whose nodes are propositions
  - *input* - have no input components,
  - *base* - have one single transition as input,
  - *view* - all the remaining.
- Connectives: **and**, **or** and **not** logic gates, and **transitions** (one-step delayed identities)
- Considered as the fastest implementation of GDL reasoner (besides compilation-based approaches)


INPUT
(does player (turnOn p))
INPUT
(does player (turnOn q))
OR
OR
VIEW
VIEW
TRANSITION
TRANSITION
BASE
(on p)
BASE
(on q)
NOT
NOT
AND
VIEW
(legal player (turnOn p))
VIEW
terminal
VIEW
(goal player 100)
VIEW
(legal player (turnOn q))

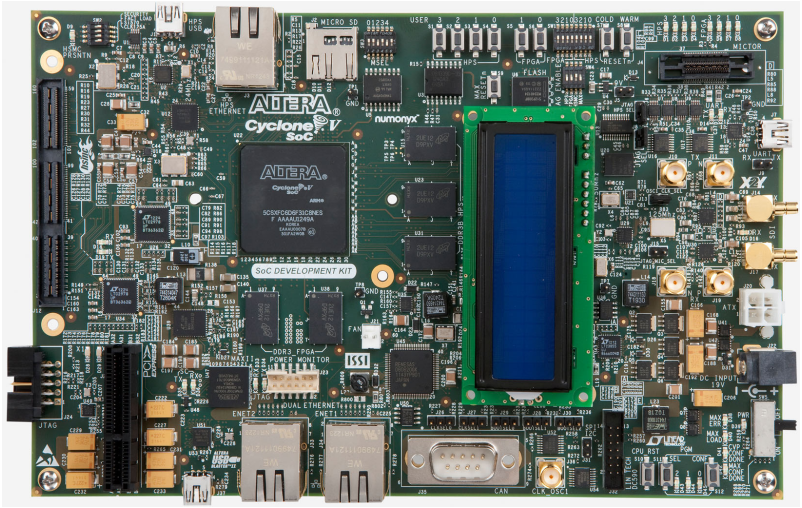
Monte Carlo Tree Search (Coulom; 2007)

- simulation-based search algorithm
- gradually builds game tree
- any-time, knowledge-free
- go, go, go!



Field-Programmable Gate Arrays

- Logic gate arrays that can be reconfigured
- Thus there is a (cheaper) alternative to manufacture hardware for your specific use
- And they can be used for prototyping
- Hardware Description Language (Verilog, VHDL) for programming
- Used in many domains: communication, image processing, control engineering, networks, cryptography, mathematics, neurocomputing, etc.
- For games: as hardware accelerators
- Great for GGP (no one tried it before) - if you can reprogram it on the fly


ALTERA
Cyclone V SoC
SoC DEVELOPMENT KIT
numonyx
5CSXFC6D6F31C8NES
HPS ETHERNET
MICRO SD
DUAL ETHERNET
ENET0
ENET1
DDR3_FPGA
POWER MONITOR
JTAG
CAN
CLK_OSC1
DC INPUT 19V
UART
HPS USB

Methodology

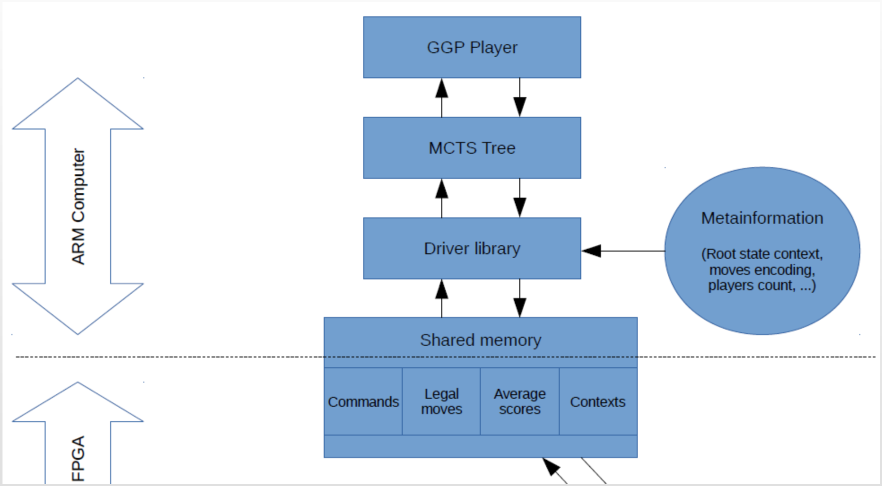
High level overview

1. Generate software propnet for a given game
2. Use predefined Verilog project template
3. Generate propnet description in HDL, compute meta-information file

- Search works on the integrated ARM computer
- Shared memory to communicate between the software and FPGA
- Player interfaces FPGA via a driver library with Java API
- FPGAs reasoner implements entire playout phase to minimize tree-to-reasoner calls
- To start playouts from a specific node FPGA has to switch context into the proper state

MCTS library

- *getRootState()*
- *getNextStates(state)*
- *getScores(state, n)* (batches instead of single playout)



GGP Player
MCTS Tree
Driver library
Metainformation
(Root state context, moves encoding, players count, ...)
Shared memory
Commands
Legal moves
Average scores
Contexts
ARM Computer
FPGA

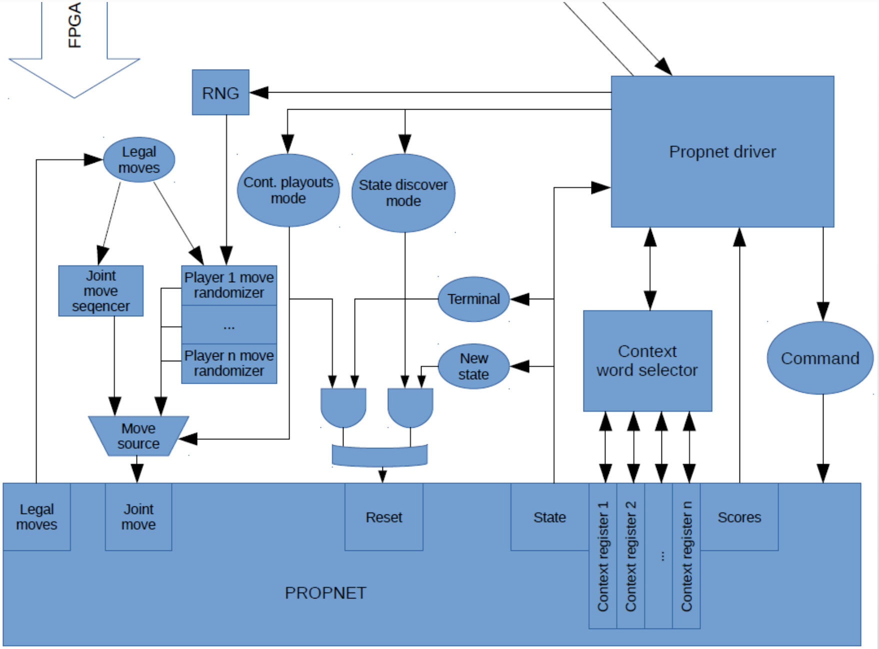
Low level overview

State representation

- Game state is coded as a bit vector - *n*th bit corresponds to the *n*th transition node.
- Division to chunks to allow information exchange through shared memory

Control modes

- *state discovery*
    - iterates over all legal joint actions
    - writes to shared memory next states + corresponding moves
- *context switching*
    - propnet driver reads a game state representation
    - writes the state to the propnet's context registers.
- *continuous playout*
    - continuously takes moves from the module generating legal random actions
    - applies them until a terminal node is reached
    - signals scores to the propnet driver and resets the internal propnet



FPGA
RNG
Propnet driver
Legal moves
Cont. playouts mode
State discover mode
Joint move seqencer
Player 1 move randomizer
...
Player n move randomizer
Terminal
New state
Context word selector
Command
Move source
Legal moves
Joint move
Reset
State
Context register 1
Context register 2
...
Context register n
Scores
PROPNET

Experiments

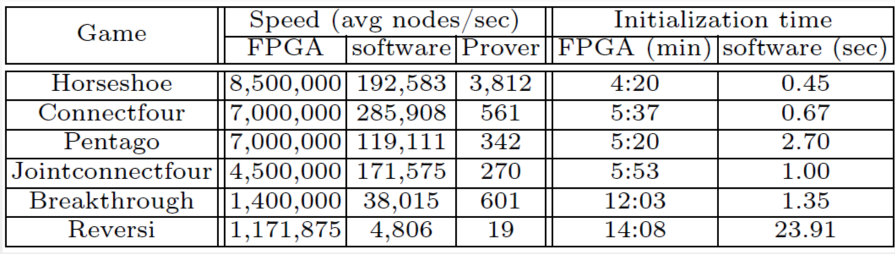
Sheer computation speed

- Flat Monte Carlo Search
- Count number of states visited in random simulations from the root
- Reasoners to compare
    - Software Propnets - the fastest propnet describe in literature (Sironi, Winands; 2017)
    - Prover - resolution engine from the GGP-base package

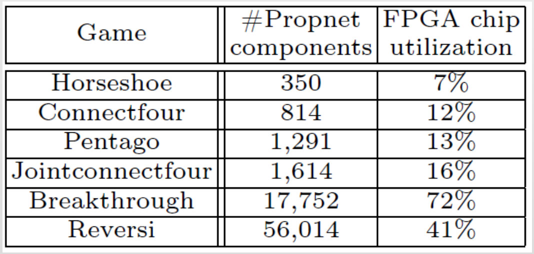
Results

- Improvement factors are between 24.5 (Connect-Four) and 58 (Pentago)
- $>$290 for Reversi (the largest propnet)
- Initialization times: 5-6 minutes (or even $>$12) instead of seconds

Computation speed comparison



Game	Speed (avg nodes/sec)			Initialization time	
	FPGA	software	Prover	FPGA (min)	software (sec)
Horseshoe	8,500,000	192,583	3,812	4:20	0.45
Connectfour	7,000,000	285,908	561	5:37	0.67
Pentago	7,000,000	119,111	342	5:20	2.70
Jointconnectfour	4,500,000	171,575	270	5:53	1.00
Breakthrough	1,400,000	38,015	601	12:03	1.35
Reversi	1,171,875	4,806	19	14:08	23.91



Game	#Propnet components	FPGA chip utilization
Horseshoe	350	7%
Connectfour	814	12%
Pentago	1,291	13%
Jointconnectfour	1,614	16%
Breakthrough	17,752	72%
Reversi	56,014	41%

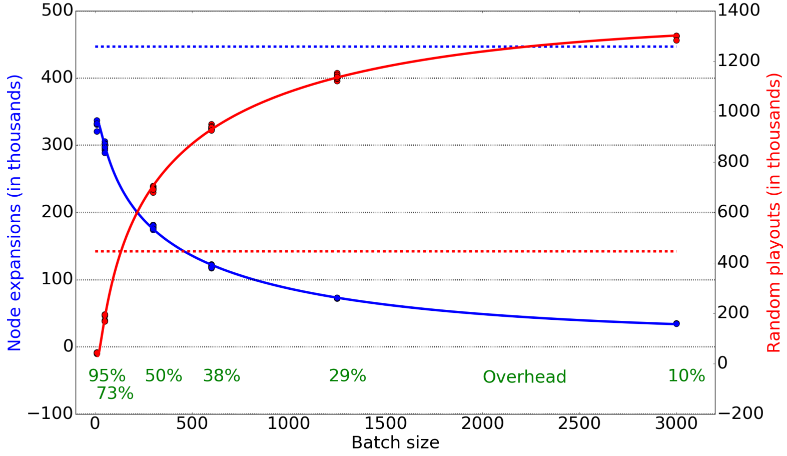
MCTS performance

- FPGAs implement playouts only,
- managing MCTS trees has to be delegated to software,
- this creates overhead – the more time spent in tree, the less number of performed simulations
- (in MCTS number of simulations straightforwardly influences the quality of the result)
- (Flat MC produces 0-overhead)
- increasing batch – makes expanded nodes more reliable, reduces overhead

Results

- FPGA-based player can perform much more playouts than baseline
- but due to the overhead it visits (opens) less nodes

MCTS performance results for Pentago



(baseline is a number of node expansions obtained by the non-batched software player)

Discussion

FPGA-based improvements

Initialization time

- Improve Verilog propnet generator module
- Support structure compilation and fitting (better hardware required)
- Propnet structure fitting (physical placement of the logic modules on a chip) – hard

Chip utilization

- Size on a chip is not 1-1 corresponding to propnet size
- Also depends on the graph planarity and synthesis toolchain optimizations.
- Largest example: breakthrough – 72%

MCTS-based improvements

Asynchronous search

- Currently reasoner is idle when tree-related phase of MCTS
- Can be fixed via scheduling tasks ahead
- Managed as multithreaded simulations from the MCTS point of view

Communication

- Optimize software MCTS library
- Use PCI-E equipped FPGA board to push ARM computer out of the loop (reduces the MCTS-reasoner communication overhead)

Summary

Justification

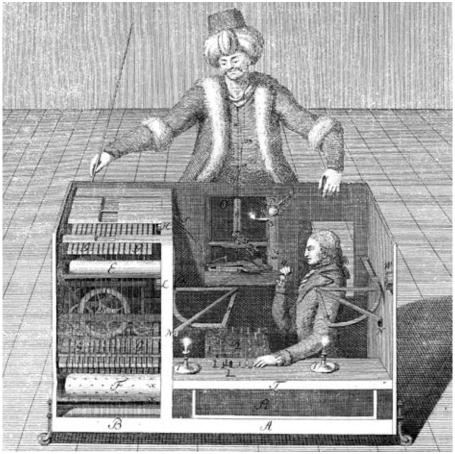
- We need fast search (as always - but in GGP particularly)
- Software-based GGP reasoners were optimized for over a decade
- This is the first approach to use hardware accelerators for the task

Contributions

- Implementing FPGA-based reasoner architecture
- Embedding it into an MCTS search
- Analyze its behavior, and identify required improvements

FPGA-based contest-ready GGP player

- Still some work to do
- Main problem is initialization time
- Basic solutions:
    - partial: knowledge-transfer – store previously generated propnets
    - general: use software propnet until the hardware one is ready



Thank you