Distributing Chess Computation for Desktop GRID

Emmanuel Jeannot, Derrick Kondo, Sangho Yi

INRIA

Mescal and Runtime Team

Emmanuel.Jeannot@inria.fr

Introduction

- RT-Boinc is a desktop grid framework that takes into account “real-time constraint”
- Playing Chess requires to take into account hard and soft deadline
- Porting a chess program to RT-Boinc is a nice way to test and evaluate it

Chess Engine

- Many chess engines are available
- Some of them are really strong
- Most of them communicate with GUIs or other environment through a standard protocol called UCI (Universal Chess Interface)

UCI chess engines

All engines (32-bit) | Free engines (32-bit) | Open source engines (32-bit)

1-2-CPU engines (32-bit) | Free 1-2-CPU engines (32-bit) | Open source 1-2-CPU engines (32-bit)

Single-CPU engines (32-bit) | Free single-CPU engines (32-bit) | Open source single-CPU engines (32-bit)

Pure lists for complete database

Pure database download

To save space, pure database has all moves stripped out, it contains PGN header and results only. This pure database is useful only for rating calculation or similar analysis, it does not have actual games, only the results.

Download pure database, 32'917 games: 0.10 MB

CCRL 40/40 Rating List — Pure all engines

Ponder off, General book (up to 12 moves), 3-4-5 piece EGTB
Time control: Equivalent to 40 moves in 40 minutes on Athlon 64 X2 4600+ (2.4 GHz)
Computed on October 1, 2010 with Bayeselo based on 32'917 games

Rank	Name	Rating ELO	+	−	Score	Average Opponent	Draws	Games	LOS
1	Rybka 4 64-bit 4CPU	3259 (−4)	+26	−25	73.1%	−154.0	44.2%	518	99.2%
2	Stockfish 1.7.1 64-bit 4CPU	3219 (+2)	+26	−25	66.0%	−103.0	49.7%	471	95.4%
3	Naum 4.2 64-bit 4CPU	3190 (+5)	+26	−26	62.7%	−83.0	49.3%	444	100.0%
4	Deep Shredder 12 64-bit OA On 4CPU	3128 (+2)	+21	−21	58.5%	−53.6	47.5%	697	99.9%
5	Zappa Mexico II 64-bit 4CPU	3082 (+8)	+21	−21	57.1%	−46.6	46.3%	695	86.3%
6	Komodo 1.2 64-bit	3066 (+10)	+21	−21	43.1%	+42.2	50.7%	698	93.0%
7	Thinker 5.4C Inert 64-bit 4CPU	3044 (+13)	+22	−21	54.5%	−30.7	38.6%	718	52.6%
8	Spark 0.4 64-bit 4CPU	3043 (+1)	+34	−35	31.5%	+122.7	43.3%	275	52.5%
9	Deep Sjeng WC2008 64-bit 4CPU	3041 (+9)	+19	−19	49.7%	+1.9	39.4%	921	93.4%
10	Hiarcs 12.1 4CPU	3018 (+9)	+23	−23	45.7%	+25.8	43.5%	605	73.6%
11	Toga II 1.4.1SE 4CPU	3008 (0)	+21	−21	54.5%	−28.6	41.4%	707	59.9%
12	Bright 0.4a 4CPU	3005 (+4)	+22	−22	53.0%	−21.5	39.8%	664	54.9%
13	Deep Fritz 10.1 4CPU	3002 (+9)	+35	−35	58.3%	−58.5	33.9%	271	99.4%
14	Loop M1-T 64-bit 4CPU	2952 (−2)	+21	−21	43.5%	+41.3	44.9%	695	79.3%
15	Crafty 23.1 64-bit 4CPU	2934 (+25)	+36	−36	51.7%	−7.5	36.9%	241	55.3%
16	Deep Junior 10 4CPU	2931 (+21)	+32	−32	50.0%	−2.7	33.5%	322	55.2%
17	Onno 1.0 64-bit	2928 (+9)	+36	−37	35.3%	+92.7	43.2%	234	100.0%
18	Spike 1.2 Turin	2843 (−5)	+16	−16	49.5%	+3.3	36.9%	1264	58.9%
19	Scorpio 2.0 4CPU	2838 (+4)	+39	−40	33.0%	+115.1	35.9%	220	54.5%
20	Delfi 5.4 2CPU	2835 (+7)	+23	−23	42.2%	+54.0	37.0%	625	57.1%
21	Bison 9.11	2832 (+5)	+32	−32	62.5%	−84.9	36.8%	315	66.3%
22	Ktulu 8	2823 (+16)	+21	−21	52.0%	−11.6	35.5%	768	51.9%
23	Twisted Logic 20080620	2822 (−1)	+27	−27	56.5%	−40.9	39.8%	440	89.6%
24	Chess Tiger 2007.1	2801 (−5)	+21	−21	53.2%	−17.9	45.7%	669	51.6%
25	SmarThink 1.10 32-bit	2800 (−22)	+39	−38	60.9%	−76.0	33.5%	224	55.1%
26	Booot 4.15.0	2797 (−1)	+25	−25	54.0%	−23.8	49.1%	479	64.4%
27	Frenzee Feb08 32-bit	2791 (−9)	+22	−22	52.2%	−14.4	39.2%	635	91.7%
28	Movei 00.8.438 (10 10 10)	2770 (−4)	+20	−20	52.4%	−16.1	40.3%	769	53.3%
29	Chessmaster 11	2769 (−1)	+28	−28	52.2%	−15.7	31.8%	424	68.1%

```
[glenan:src/Chess/rtboinc-chess] ejeannot% stockfish-171/Mac/stockfish-171-64-ja
Stockfish 1.7.1. By Tord Romstad, Marco Costalba, Joona Kiiski.
go depth 10
info depth 1
info depth 1 score cp 72 time 64 nodes 20 nps 312 pv g1f3
info depth 2
info depth 2 score cp 12 time 97 nodes 47 nps 484 pv g1f3 g8f6
info depth 3
info depth 3 score cp 68 time 97 nodes 146 nps 1505 pv g1f3 g8f6 b1c3
info depth 4
info depth 4 score cp 12 time 98 nodes 290 nps 2959 pv g1f3 g8f6 b1c3 b8c6
info depth 5
info depth 5 score cp 28 time 98 nodes 613 nps 6255 pv g1f3 g8f6 b1c3 b8c6 d2d4
info depth 6
info depth 6 score cp 12 time 100 nodes 1067 nps 10670 pv g1f3 g8f6 b1c3 b8c6 d2d4 d7d5
info depth 7
info depth 7 score cp 32 time 118 nodes 2188 nps 18542 pv g1f3 g8f6 b1c3 b8c6 d2d4 d7d5 c1f4
info depth 8
info depth 8 score cp 12 time 121 nodes 4508 nps 37256 pv g1f3 g8f6 b1c3 b8c6 d2d4 d7d5 c1f4 c8f5
info depth 9
info depth 9 score cp 12 time 144 nodes 10297 nps 71506 pv g1f3 g8f6 b1c3 d7d5 d2d4 b8c6 d1d3 e7e6 e2e4
info depth 9 score cp 28 time 165 nodes 27298 nps 165442 pv e2e4 g8f6 e4e5 f6d5 c2c4 d5b4 d2d4 b8c6 b1c3
info depth 10
info depth 10 score cp 16 time 180 nodes 35334 nps 196300 pv e2e4 g8f6 e4e5 f6d5 c2c4 d5b4 d2d4 d7d6 a2a3 b4c6
info nodes 51775 nps 260175 time 199 hashfull 0
bestmove e2e4 ponder g8f6
```

http://www.computerchess.org.uk/ccrl/4040/rating_list_pure.html

Stockfish

- Open-source
- Portable (Mac/Linux/Windows)
- Multithreaded
- Very strong
- Comes with a opening book

Chess AI

Basically:

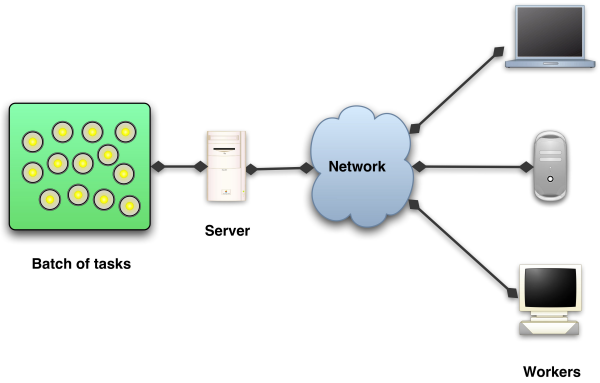
- Tree search (each node being a position each edge being a move).
- Evaluation function of position
- Pruning techniques to accelerate search (min/max alpha/beta). Up to 1200kn/s.
- Time management
- Opening book

Chess on Desktop Grid

Constraints:

- No communications between workers
- Minimal communication between workers and server (task and result)
- Potentially many workers but not always
- Churn
- Some workers may never return an answer
- No time to generate a different task for each work

Impossible to use min/max algorithm in this context



Problem

The server has a position.

It needs RT-Boinc to compute the best possible move in a given amount of time

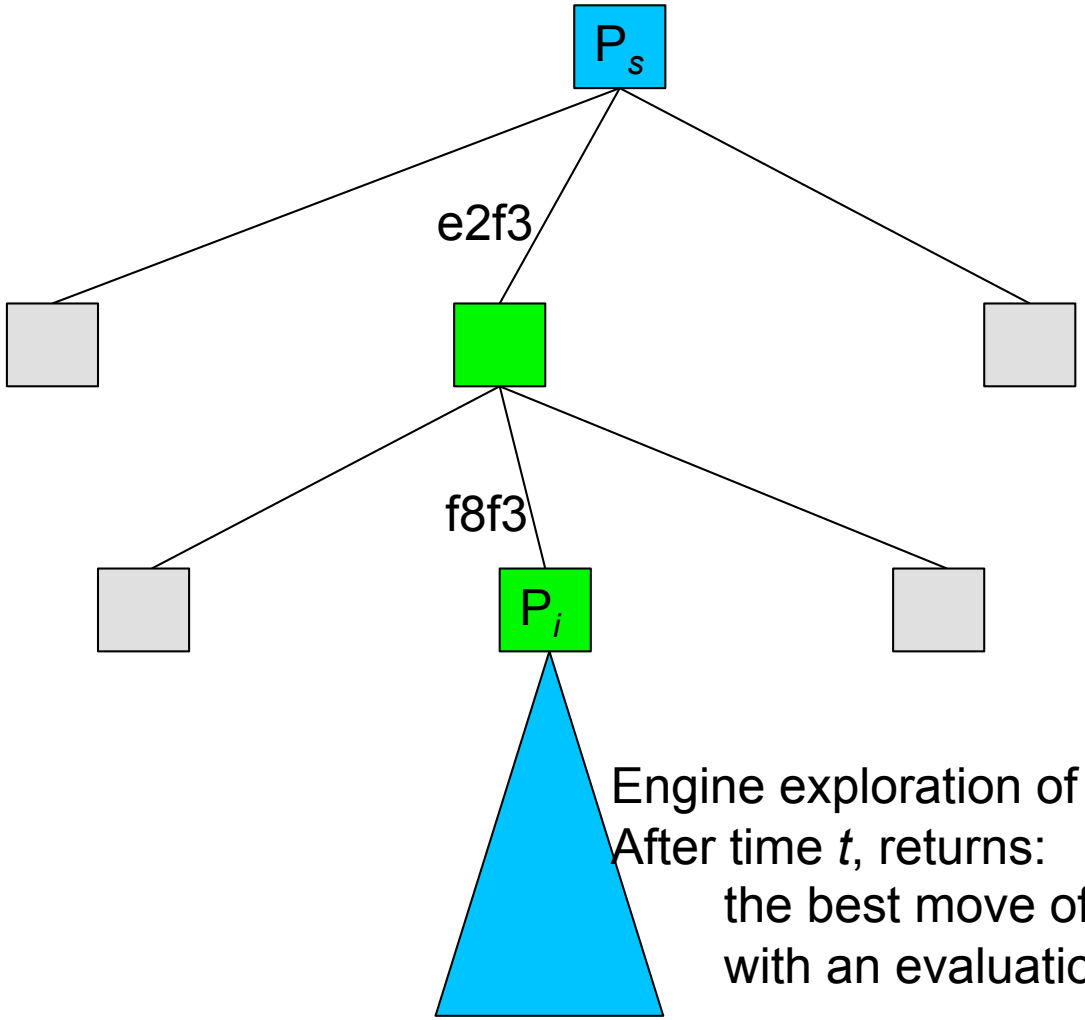
How to distribute the search of the tree among the workers

Solution: a randomized algorithm

Principle:

- Each worker receives:
  - the same position P_s,
  - an integer *n* and,
  - a soft real constraint ***t*.**
- Each worker *i* plays *n* moves at *random*: reach position P_i
- Each worker *i* asks the chess engine to process P_i until time *t*
- At time *t*, the worker returns the best move found by the chess engine with an evaluation and the *n* moves required to reach P_i
- The server aggregates clients results using min/max algorithm and compute the best result for position P_s with its evaluation

Example (n=2)



The workers returns:

- P_s
- e2f3 ; f8f3
- P_i
- g2f3
- -4.5

Engine exploration of the subtree from P_i
After time *t*, returns:
the best move of P_i (e.g. g2f3)
with an evaluation (e.g. -4.5)

Server aggregation

Get all workers answer

Aggregates answers to built a tree *T'*

Apply min/max on *T'* to determine the best move

Question: how far is T' from the real tree T?

Uniform Random Choose

How many workers are required such that almost surely all the nodes of depth n will be considered?

Hits: the average arity of a chess tree is 30.

At depth n we have $m=30^n$ nodes. If we assume that each node is selected uniformly.

Formally, I have m coins in a bag, I pick one uniformly and replace it. How many picks do I need such that there is probability ε that I did not pick all of them at least once.

Uniform Random Choice (Cnt.)

Approximation of the solution:

1/m: probability that a node (a position) is explored

1-1/m: probability that a node is not explored

$(1-1/m)^p$: probability that a node is not explored after p trials

$1-(1-1/m)^p$: probability that a node is explored after p trials

$(1-(1-1/m)^p)^m$: probability that all nodes are explored after p trials

Example: n=3, m=27000, ε=0.01

$(1-(1-1/27000)^p)^{27000}=0.99$, $p\approx400000$

n=2, m=300, ε=0.01

$(1-(1-1/300)^p)^{300}=0.99$, $p\approx4450$

Towards a Bias Random Choice

Uniform random choice:

- Requires many workers to be sure that we did not miss a good move

Not all choices are equivalent

Need to bias the search towards “the best” moves.

Question: how to rapidly estimate what are the best choices?

Best Moves Fast Estimation

Evaluating a move at a fixed shallow depth (5), is fast (orders of ms).

How good is this evaluation?

In general it appears to be good but this is not always the case.

Evaluating the Quality of Depth 5 Evaluation

Question:

- Given a position
- The best move for this position
- What was the rank of this move at depth 5, among all possible moves?

Experiment:

- We took 1407 positions from the (all games of 1972 Spassky-Fischer world championship match, removed opening book positions)
- Most of them are tight positions (21 games, 11 draws)
- The engine analyzes each position for 20 minutes
- We record the rank of the best move when the search reaches depth 5

Results

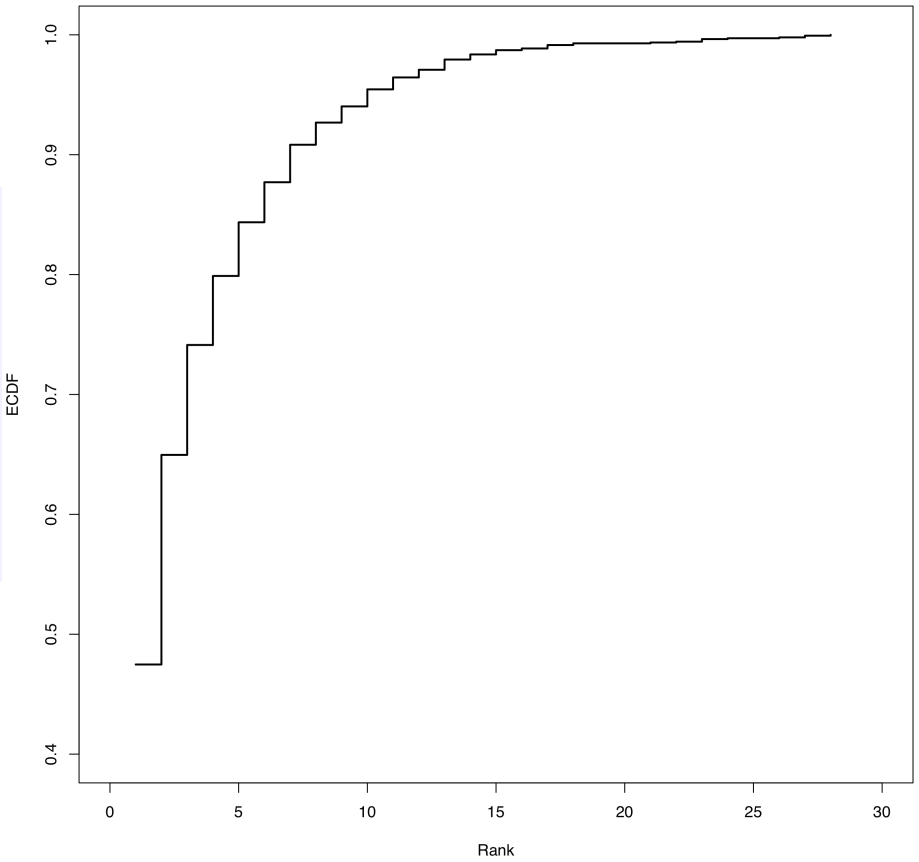
47.47% of the “best” moves are ranked 1 at depth 5

17.48% of the “best” moves are ranked 2 at depth 5

9.17% of the “best” moves are ranked 3 at depth 5

…

0.07% of the “best” moves are ranked 28 at depth 5



Biased Random Choice

Method:

- For each position
- Enumerate each possible moves
- Rank them according to the depth-5 estimation
- A move is chosen according to the empirical law shown before (N°1 with 47.47%, N°2 with 17.48% etc.)

Advantage:

- The more likely a move is to be good the more chance it has to be chosen
- Redundancy of good position (tolerance to churn and failure)

Conclusion

Distributed mechanism for chess computation

Random algorithm on the client side for scalability and reliability

Bias random choice based on empirical analysis of positions

Todo:

- Evaluation of the model
- Improved model (taking into account the arity of each node)
- Taking into account the heterogeneity of workers when doing the aggregation