Harnessing Human Computation Cycles for the FPGA Placement Problem

L. Terry, V. Roitch, S. Tufail, K. Singh, O. Taraq, W. Luk, and P. Jamieson
Department of Computing, Imperial College, London, United Kingdom

Abstract

Harnessing human computation is an approach to find problem solutions. In this paper, we investigate harnessing this human computation for a Field Programmable Gate Array (FPGA) placement problem. We create a game called Plummings. In this game, a player attempts to reduce the critical path of a digital design mapped to an FPGA by swapping clusters on the array, but the problem details are abstracted away, and instead, the game simply presents a challenging problem where paths must be minimised to save the game characters - the Plummings. Once players have played a level, the placement is can be evaluated in VPR. Our results show that 4 human players over a set of 5 benchmarks can create placement solutions with comparable critical paths compared to VPR's solutions. This is not always the case, and we suggest some reasons and further approaches to improving our results.

Keywords: FPGA, Placement, Harnessing Human Computation

1. Introduction

As the video game market continues to dominate the entertainment market over previous leaders such as movies and music, individuals are spending their free time playing video games of all sorts. This relaxation time is spent solving challenges designed into a game world for the players, and the human, even though in a relaxation state, is performing computation. This computation is wasted in terms of what society might consider useful work. If, however, we could present challenging problems that a player considers fun to solve, but these problems and solutions can be applied to real-world problems, then we would have a win-win situation. For any optimisation problem, extending the time dedicated to searching the solution space potentially improves results. In this work, we focus on harnessing what would be wasted human computation time for searching this space.

This concept of harnessing human computation has been explored in games such as Foldit [1] and Google's ESP game [2]. In this paper, we apply this concept of harnessing human computation to attempt to improve the FPGA placement results for a digital design. In contrast to the previous experiments, this work begins to pose the question if harnessed human computation can be useful in the domain of classical optimisation problems.

The FPGA placement problem consists of determining how to place a set of logic clusters on an array of Field Programmable Gate Array (FPGA) tiles such that the critical path of the circuit is minimised. A common FPGA exploration tool, VPR 5.0 [3], uses a simulated annealing algorithm to place circuits on FPGAs. In this work, we use VPR 5.0 to generate placement files that are then loaded into our video game environment. The game, called Plummings, allows a player to freely swap clusters on the array; however, details of cluster swapping and the critical path are abstracted away from the player in the form of a game. Each swap will either improve or degrade the resulting speed of the circuit based on the critical path length. This is reflected in a score, and players attempt to improve the critical path.

Plummings the game does not present the technical aspects of FPGA placement to the user and the complexities of digital designs. Instead, the game shows your score as the current critical path improved relative to the initial critical pressure in the form of an air pressure score. Nodes on this path can be moved to reduce the path and save the beings in the game world, Plumms, by increasing the air pressure and providing them with adequate oxygen. This abstraction harnesses the human computation by presenting the problem as a fun and challenging game as opposed to work. Additionally, we have created a competitive multiplayer environment that pushes players to improve the placement even further in the disguise of the challenge to beat another human opponent.

Our preliminary results show that for 5 existing benchmarks and 4 game players the final placed and routed designs were on par, and in one case better, than the critical paths generated by 5 random seed placements by the simulated annealer in the VPR tool. This suggests that with an

increase in the number of players and improvements to our critical path estimation tool this approach may be a feasible application of recycling wasted human computation.

The remainder of this paper is organised as follows. Section 2 describes previous work in the field of harnessing human computation, the basic structure of an FPGA relevant to the placement problem, and details of the placement problem. Section 3 describes the game Plummings and how we abstract the placement problem to a game form including the estimation of critical path. Section 4 shows our results of placement solutions based on the playing of the game compared to VPR's results. Finally, Section 5 concludes the paper and discusses some future work.

2. Background

In this Section, we review previous work in the field of harnessing human computation. In our case, human computation is harnessed to solve the FPGA placement problem, and for this reason, we will introduce aspects of FPGA architecture and a CAD flow for these devices.

2.1 Harnessing Human Computation

Humans and computers perform computation where both are limited in what they can achieve due to a number of factors including time. Harnessing human computation aims to utilise so called "wasted cycles" (where a person is not trying to do useful work) to perform useful computational tasks.

Existing applications that harness human computation to solve different tasks include projects such as protein folding (FoldIt [1]), picture identification (Google Image Labeler [2]) and recommendation (Amazon and Apple Genius [4]), which we briefly review here.

FoldIt [1] is a game developed to allow players to interact with complex proteins, attempting to solve one of the problems facing biological science - how to predict protein structure. FoldIt presents 3D abstraction of proteins and simplifies the chemistry behind protein folding by making the problem into a game. It is clear to the user that they are folding proteins as no abstraction is attempted, but users are rewarded for their efforts by a driving goal to get to the top of the leader board. FoldIt develops solutions that are better than those produced by machines by encouraging competition between the 50,000+ players of the game and has been successful.

An ongoing problem in the realm of computer vision is identifying and labelling objects in an image. This problem is simple for humans to solve, but challenging for machines based on current technology advances. The Google Image Labeler (based on the ESP game [2]) tackles this image tagging problem by turning a menial task, for humans, into a game. The game is based on cooperative play between two players. Each player types what they believe describes the presented image and the aim of the game is for the two players to agree on the description by only communicating through the game interface. Players are driven to score well by matching words and at the same time help solve the tagging problem. The results obtained from the original version of the game show that 5,000 players could label every image in Google image search in approximately two months. This application of human computation has been so successful that it's been developed further into locating objects in images in a project called Peekaboom [5].

The designers of the image tagging game showed that it is possible to make people want to play games that have real-world applications. Furthermore this is a fast and cheap way of producing solutions to difficult problems, Internet gaming sites have millions of unique users every month [6], which is synonymous to having millions of computers.

Another popular application of harnessing human cycles has been incorporated in CAPTCHAs - Completely Automated Public Turing test to tell Computers and Humans Apart. CAPTCHAs are popularly used on web forms where a user sees an image of text and symbols and has to type in the sequence to prove they are a human as opposed to a bot. Ahn *et. al.* leverage this text identification process to use humans to identify archived printed text into computer format [7].

Amazon recommendations and the Apple iTunes Genius service silently collect information about people's likes and dislikes. In general, people who like the same song/product will have similar tastes. This is human computation in a more subtle light where the people browsing Amazon or listening to iTunes are grouping together similar items (whether they are related products or of similar styles) they give semantic meaning to products, which is very difficult for a computer to achieve.

The goal of this work is to explore these concepts of harnessing human computation and apply it to the FPGA placement problem as described below.

2.2 FPGA Architecture

FPGAs are programmable chips that can implement any digital design. Figure 1 shows the basic structure of an island style FPGA consisting of an array of logic blocks and I/O cells that are interconnected using programmable routing [8]. The programmable routing consists of wire segments that are connected to logic blocks and other wire segments via programmable switches at switch blocks and connection blocks. The logic blocks are also called clusters where these clusters commonly consist of a combination of Look-up Tables (LUTs), flip-flops, and internal pro-

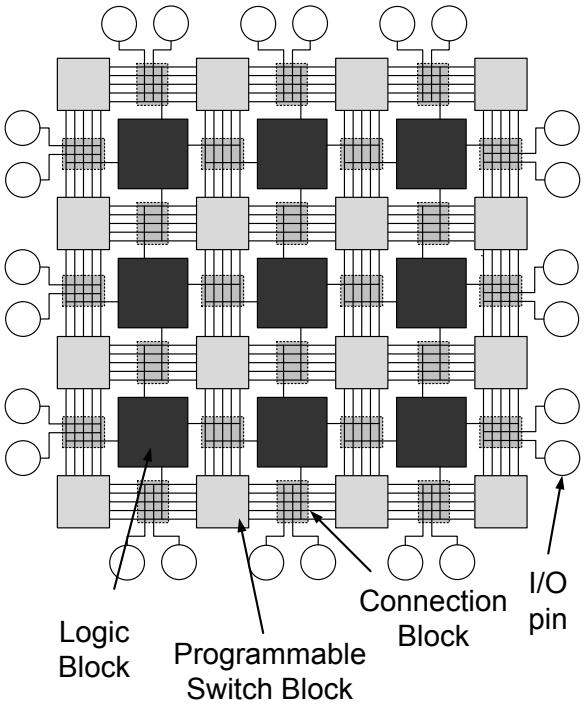

Fig. 1. *An FPGA consisting of logic blocks, I/O pads, and programmable routing.*

Table 1. FPGA Architectural Parameters Used in Paper

Parameter	W	N	K	F_{cin}	F_{cout}	F_s	L
Value	180	10	4	0.17	0.1	3	4

grammable routing. From the placement problem perspective, the most important architectural feature is the cluster, and the reader needs to understand that the FPGA consists of an array of these clusters that a design will then be mapped to.

Digital designs are mapped to these devices using a CAD flow similar to the one shown in Figure 2. In this figure a high-level design, normally written in a Hardware Description Language such as VHDL [9] or Verilog [10], is converted first into logic gates, then LUTs, and then clusters, and finally, these clusters are placed and connections between them are routed.

For the soft logic fabric we use to illustrate the concepts in this paper, we will select a single set of parameters chosen to be close to the typical parameters of a modern FPGA, including the use of direct-drive (also known as uni-directional) routing [11]. The parameters we use in this paper are given in Table 1 and for more details about these parameters see [8].

The focus of this work is on the Placement phase of the CAD flow, which we describe in more detail.

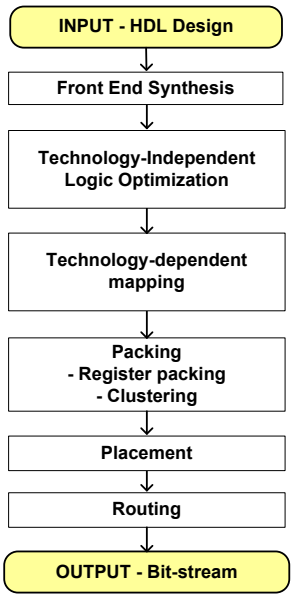

Fig. 2. *A graphical representation of CAD flow for FPGA design.*

2.3 FPGA Placement Problem and Simulated Annealing

The goal of the placement problem for FPGAs is to place the clusters, representing the digital design, onto the array of FPGA clusters such that the critical path (the longest path from either a primary input to a primary output, a primary input to flip-flop, flip-flop to flip-flop, or flip-flop to primary output) is minimised. This problem has been shown to be NP-complete to solve optimally, and a number of popular algorithms have been used including the focus of this paper, simulated annealing [12].

The simulated annealing algorithm, when used in the FPGA domain for placement, attempts to minimize the critical path based on the process of cooling metals. A cooling schedule controls the weighting of a probability function. This function determines if randomly selected swaps between clusters on the FPGA are accepted or not. Each random swap of clusters will either improve or degrade the critical path, and initially, all swaps are accepted regardless if they improve the critical path or not. As the temperature cools, only swaps that improve the critical path are accepted. In this way, the early phases of the cooling schedule is used to allow hill climbing that will, hopefully, avoid local minimums in this optimisation problem.

VPR 5.0 [3] is a tool used for FPGA architecture exploration, and this tool implements simulated annealing as the placement algorithm. A netlist of clusters is inputted to the tool. This netlist is read in and placed and routed

onto an FPGA architecture, which is described by a set of architectural parameters. These parameters describe the size of the clusters, LUTs, and the connectivity of the programmable routing. VPR outputs the placement coordinates of the design's clusters, the routing resources used to connect the clusters in the programmable routing, and the area and speed of final mapped design to the described FPGA architecture.

In this work, we use the VPR 5.0 tool to both create initial placements for our game abstraction of the placement problem and then to evaluate our human placed results compared to results generated within the VPR framework. The results generated by VPR depend on an initial placement of clusters, which is randomly generated. This randomness is controlled by a random seed that is also inputted into the tool.

3. System Description

In terms of mapping the placement problem to a game, there are two parts to consider. First, calculating an estimation of the critical paths between a network of clusters and programmable paths through the FPGA to represent this as a score, and second, modifying the placement to reduce the critical path and near critical paths via player actions. As such we have developed two separate elements to our game, Plummings. The calculation of critical paths through the FPGA is performed in an API that abstracts this process so that it can be modified without changing the game.

In this Section, we will describe the game, show how the game is an abstraction of the placement problem for FPGAs, and describe some details of our critical path estimation.

3.1 The Game Environment

Our particular game is set in the Plummings world as can be partially seen in Figure 3. This world is inhabited by characters known as Plumms. Each Plumming colony is a complex network of pipes joined by cogs on a two dimensional grid. Each colony (game level) has a limited supply of oxygen coming into the pipes, and this is not enough for all the Plumms to survive. A player attempts to minimize the network of pipes to increase the pressure, and therefore, supply enough Oxygen to the Plumms.

In terms of players achieving this, the Plummings board consists of a grid showing all the possible spots pipes are joined. A player sees one collection of pipes displayed. This does not show the full complexity of the colony. Joining each a set of pipes is a cog that can be selected with a mouse click and dragged to a new spot.

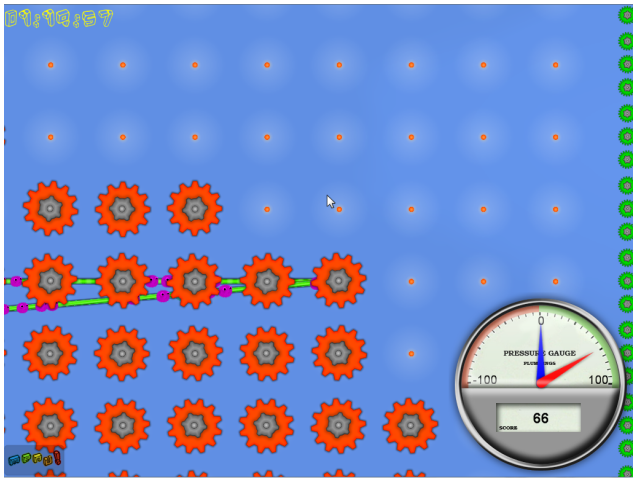

Fig. 3. *The Multi-player View of the Plummings World.*

In the game, a selected cog when moved to other potential grid points will either display a yellow circle or a green check mark at the mouse pointer. The yellow circle indicates that this grid location is currently occupied with another cog and letting go of the mouse will be swap the existing cog with the dragged cog. A green check mark on the mouse pointer, however, means that this spot on the grid is empty and the cog can be moved here without performing a swap with another cog.

Once a player has swapped or moved a cog to a new spot, the player will either be shown a new collection of pipes that is causing the Plummings problems or this same path will remain as the problem path. At the bottom of the screen there is an undo and redo button to allow the player to move forward and backwards in swaps so that there is no concern about the consequences of making a particular move.

3.2 FPGA Placement to Game Abstraction

As projects related to other industries have shown [1], [2], problems formulated into a game can capture the attention of many thousands of players. Ideally, the game should be attractive, fun, and addictive. Part of what makes a game playable and fun is interaction with other players, it is essential to encourage competition in games with a purpose (i.e. make people want to do better) otherwise there will be no drive to improve existing solutions.

In terms of the FPGA placement abstraction, the pipes displayed in the game map directly to edges (or wire connections) between clusters and the cogs represent the clusters themselves. The current path displayed in the game shows the present critical path for the design, as measured by the Manhattan Distance, for the current FPGA

placement, and the pressure gauge shows a score of how successful the player has been in improving the circuit's original critical path compared to the current critical path.

We create a "score" for the user based on the current critical path of the user's placement. The user then tries to maximize their score by minimising the critical path. The process of minimising is achieved by moving two connected nodes in the critical path closer together. If a player places a node onto a cluster that is occupied, then the nodes are silently swapped. Moving a node far from its original location, in effect, swaps it with a node far away, which may have adverse effects on paths running through the swapped node and therefore reduce the score.

This process of swapping is done at the players discretion without much additional information to make an informed decision on which cluster to choose and where to move it. Since the player only sees the current critical path, the choice of cluster swaps is made mainly to reduce this critical path. The game still allows the human to explore more of the optimisation search space, but this search is neither truly random nor is it made with informative information. In future, we would like to provide additional information to the user and a choice of paths to attempt to shorten.

Each circuit is initially placed by VPR. This initial placement is used to calculate the base score for players to improve on. We could remove this requirement and instead start with an unplaced circuit, however, it would take far more time for a player to reach a reasonable score. This initial placement is also a point where it is possible to conceive some function that allows users to pick from a set of current best placements. This would lead to incremental improvements over the best placement, but could lead to the optimisation being trapped in a local minimum.

We use each placement of a particular benchmark as a starting point for a level within the game environment. Players can select from a set of levels/circuits, play the game for some amount of time, and then unlock further levels/circuits. The environment maintains the best placement so far for the user, and emails the placed results to a server so that we can collect the best placement results.

The placed file is converted into an object (representing the circuit) used by the API and stored in this form. When a game is selected the API loads up the stored placement object and the player can begin modifying it. The API only calculates a new score when a modification to the placement is made, via specific API calls such as swap() or move().

The game is, from our small player sample, preferred by people in multi-player mode. In this mode, a player sees what is the current best global score for other players in real time, and tries to beat that score in a given amount of time. In figure 3, the pressure gauge, in the right bottom corner, shows the difference between two players. The two needles show both the players and opponents relative score.

The popularity of the multi-player mode relies on the fact that in general gamers aim to beat themselves and every other player, this is the same motive exploited by FoldIt. Similar to FoldIt, we specify a threshold value that we class as a 'pass' value, resulting in the player receiving skill points. In this way more challenging circuits can be locked until a player has enough skill (experience) to attempt the harder circuits.

With all game design, play testing is a key stage in which designers determine how to make the game the most fun. From our limited experience, we have found that the multi-player game is the best in terms of motivating players to compete and find the best placement. This, however, may change if we had a large user base of players and some form of leader board as done in FoldIt.

3.3 Critical Path Calculations

In terms of calculating present critical path for the current score, the most accurate method would be to use VPR's routing tool and calculating the exact critical path for each placement iteration. This, however, is not feasible in a real-time game such as Plummings as the router takes significant time. For this reason, we have implemented an API that estimates the critical path, understanding that the loss in accuracy may affect our final results.

To calculate a new score the API employs a Critical Path Method (CPM) algorithm. This requires a directed, acyclic graph that is obtained by looking internally at the clusters and treating flip-flops as primary inputs and primary outputs. The CPM involves the following stages:

1) Forward Pass - Walk the graph from inputs to outputs and record, at each node, the maximum cost of getting from the start node to the current.
2) Backward Pass - Walk the graph from outputs to inputs and record, at each node, the minimum cost of getting from the start node to the current.
3) Subtract Values - Subtract the Forward Pass values from the Backward Pass values.
4) Collect Critical Paths - Walk the graph forwards, following the nodes which have value 0 (slack = 0).

As the score needs to be calculated at every move we need to ensure that the algorithm is fast. We achieve this with parallelization:

1) Split the circuit into 4 circuits, one for each type of connection (input-output, input-flip-flop, flip-flop-flip-flop, flip-flop-output).
2) Invert each of the 4 circuits (to compute the backward pass).
3) Execute the CPM passes concurrently on each of the circuits (8 in total - 4 forward passes, 4 backward

passes).

4) Once the threads are completed, combine the results to calculate the critical paths.
5) Localised updating by not recalculating the path calculations if the path is unaffected by a move.

This approach to calculating the Manhattan distance is not novel, but is an important consideration given the game abstraction that we are attempting to provide to the player. Our approach needs to deal with calculating critical paths in a closer to real-time speed to allow for the game to be played by a human. The placer within VPR performs similar estimations of critical path, but in our context we have more flexibility in trading off accuracy for speed of the calculations. This is another dimension of the problem, which we have not had time to explore further.

4. Results

To study the success of our approach, we conduct a small study on 5 different benchmarks of varying sizes that come from the MCNC benchmarks [13] and OpenCores [14]. We compare the placement results of 4 different human players against 5 different random seeded VPR placements. By comparing the solutions produced by VPR and human players of our game, Plummings, we can see how harnessing human computation performs in the FPGA placement problem.

For these experiments, we have fixed the architecture parameters of the FPGA. For the interested architects, we use 4 input LUTs, clusters which include 10 LUTs, and the routing architecture uses default parameters that come with the distribution of VPR 5.0. The benchmarks are mapped to VPR 5.0 using the same CAD flow as presented in [3].

Figure 4 shows the five benchmarks critical paths as determined by VPR after being placed by either players or VPR using different random seeds. Critical path is displayed on the y-axis and the benchmarks are listed across the x-axis. For each benchmark, the first five bars are for the placements by VPR for different random seeds. The next two bars show the geometric averages of first the VPR placements and then the user placements respectively. Finally, the next four bars are the critical paths by those of the 4 humans. The shorter the bar means the shorter the critical path, which means the circuit will operate faster.

In 4 of the 5 benchmarks, the human players are doing as well as the computer. The worse case was for "mac2" where players are performing significantly worse than VPR. Our initial thoughts are that this benchmark, which is an implementation of a multiplier accumulator, there are a high number of near critical paths due to the structure of the multiplier. Our game abstraction, which presents critical paths first, is a greedy presentation to the user and in a way does not pass on information about the large number of similar paths that will cause problems. In this case there are too many near critical paths for our approach to deal with successfully.

We also see that some of the poor placements are due to the course modelling of critical path lengths. In all cases, players are able to improve the in game "score" presented to them, but this does not necessarily translate into improvements once mapped to VPR. Our model lacks knowledge of routing behaviour, and we cannot examine these cases and update the scoring as appropriate, however, changes to the model could improve the scoring, and therefore, help move players in the right direction.

The results also show that for the "clma" benchmark, one user found an exceptional solution (20% improvement from the averaged VPR critical path) that is superior to all the other placement results. This result suggests that given a critical mass of players that these exceptional solutions may be found more often. The argument to this success is that given enough time, computing machines and humans will find such solutions as more of the search space is explored. This is true, but considering that a human will be playing a game regardless of work, harnessing that time to find better solutions to an optimisation problem comes for free as opposed to the computation and energy resources used by computational machines dedicated to the task.

5. Conclusion and Future Work

In this paper, we have created a game that can be used to abstract the FPGA placement problem in order to harness human computation in a fun and entertaining way. Furthermore, we have provided preliminary evidence that humans can produce results as good as those produced by algorithmic solutions, and in some cases will find exceptional solutions due to longer exploration of the search space. For five benchmarks and a small sample of human players, our results show that humans were successful in matching results from the VPR tool, and in some cases finding significantly better solutions. This comes at no additional cost since harnessing of human computation is recycling so called "wasted cycles".

The bigger question of applying harnessing techniques to general optimisation problems suggests that humans, if presented a problem in greedy form, may randomly find better solutions. This is due to the increased time exploring the search space.

To play a Vista demo version of the game, download at `http://www.doc.ic.ac.uk/~pjamieso/plummings/Plummings.msi`.

From a perspective of improving our game abstraction for the placement problem, it would be useful to make critical path calculations more accurate approximations to the routing results obtained through VPR. This problem,

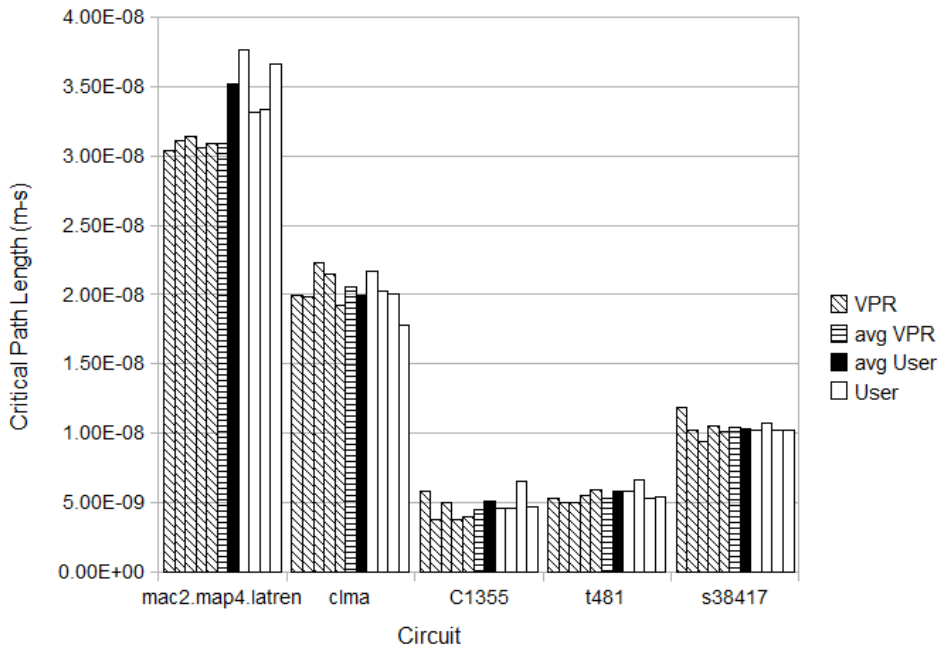

Fig. 4. Bar Graph showing the final Critical Path as calculated by VPR for players versus VPR placement with different random seeds.

however, exists in all stages of FPGA CAD flow where there is a desire for low-level detailed information to be accessible at higher points in the CAD flow. One approach may be to occasionally run VPR's routing algorithm during the playing of the game to obtain more accurate low-level information.

With better models, we believe a more detailed study is needed where our game would be distributed on the web to reach a large audience from which we can obtain large amounts of data to compare to VPR results. Research into human computation by exploiting gamers has revealed that with the right game, many thousands of people are willing to play, making a huge computational resource available to solve new problems.

In terms of optimisation problems, our poor results on larger circuits suggests that it might be appropriate to break the problem up and distribute these problems, abstracted into games, to a number of people. This divide and conquer approach, however, suffers from challenges of local optimisations versus global optimisations much like the algorithmic counterparts. In addition, there are a number of strategies that may be useful to explore, and it would be interesting to analyse human solutions to see if there is any knowledge generated by our own heuristic approaches.

References

[1] "FoldIt http://fold.it/portal/," 2008.

[2] "ESP Game http://www.gwap.com/gwap/gamesPreview/espgame/," 2008.

[3] J. L. Ian Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose, "VPR 5.0: FPGA CAD and Architecture xploration Tools with ãĆůngle-Driver Routing, Heterogeneity and Process Scaling," in *ACM/SIGDA International Symposium on FPGAs*, Feb 2009.

[4] "Apple Genius http://www.apple.com/itunes/features/#genius," 2008.

[5] L. von Ahn, R. Liu, and M. Blum, "Peekaboom: A Game for Locating Objects in Images," in *ACM Conference on Human Factors in Computing Systems*, 2006, pp. 55–64.

[6] "Miniclip http://corporate.miniclip.com/about-miniclip.htm," 2008.

[7] Luis von Ahn and Ben Maurer and Colin McMillen and David Abraham and Manuel Blum, "reCAPTCHA: Human-Based Character Recognition via Web Security Measures," *Science*, pp. 1465–1468, Sept. 2008.

[8] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[9] *IEEE Standard VHDL Language Reference Manual*, IEEE, 1987.

[10] *Verilog Hardware Description Reference*, Open Verilog International, March 1993.

[11] G. Lemieux and D. Lewis, "Directional and Single-Driver Wires in FPGA Interconnect," in *IEEE International Conference on Field-Programmable Technology*, Dec 2004, pp. 41–48.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[13] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," 1991, tech. Report. Microelectronics Centre of North Carolina. P.O. Box 12889, Research Triangle Park, NC 27709 USA.

[14] "http://www.opencores.org," 2007.