Human Computation (2015) 2:1:69-103

ISSN: 2330-8001, DOI: 10.15346/hc.v2i1.5

Crowdsourcing the mapping problem for design space exploration of custom reconfigurable architecture designs

ANIL KUMAR SISTLA, UNIVERSITY OF NORTH TEXAS

KRUNALKUMAR PATEL, UNIVERSITY OF NORTH TEXAS

GAYATRI MEHTA, UNIVERSITY OF NORTH TEXAS

ABSTRACT

One of the grand challenges in the design of portable/wearable electronics is to achieve optimal efficiency and flexibility in a tiny low power package. Coarse grained reconfigurable architectures (CGRAs) hold great promise for low power, high performance, and flexible designs for a domain of applications. CGRAs are very promising due to the ability to highly customize such architectures to an application domain. However, greater customization makes the mapping of applications onto these architectures very challenging. Good tools and fast, effective mapping algorithms are needed to support design space exploration for CGRAs. In particular, the mapping problem has been difficult to solve in a satisfying and general way. In this paper, we present an architectural design flow using crowdsourcing to provide mappings of benchmarks onto new architectures. We show that the crowd can provide high quality, reliable mappings, significantly outperforming our custom Simulated Annealing algorithm in almost all cases. We further show that the crowd can provide other types of feedback that are difficult to obtain from an automatic mapping algorithm. Our proof of concept cross-architectural study supports an 8Way or 4Way1Hop architecture as a top choice, concludes that a custom modification that constrains inputs and outputs consumes less energy but requires more area than its less constrained counterpart, and suggests that Stripe architectures are interesting to consider because they perform nearly as well as our mesh variants and may present a more straightforward mapping problem for the crowd or an automatic mapping algorithm.

1. INTRODUCTION

The portable/wearable device market is growing exponentially, with consumers demanding higher speeds, longer battery life, and instant access to anything, anytime, anywhere. Benefits of such devices for quality of life are only beginning to be felt in critical areas from health care to safety and security to personal multimedia to aerospace.

Portable/wearable devices have many competing design goals - they must be light-weight, mobile, flexible, low-power, and high performance. The chip architecture that is selected is a primary determinant of how well a device meets these competing design goals. First of all, the architecture must be reconfigurable; it must be capable of running many different applications within a domain. Second, the architecture should be customized as much as possible so that it can run the intended applications with great efficiency and low power consumption. Field Programmable Gate Arrays, for example, while reconfigurable, are more general purpose and may be highly inefficient. Coarse-grained reconfigurable architectures (CGRAs) are an intriguing spot in the design space. They are reconfigurable and highly customizable. A typical use case for CGRAs today would be to run computational intensive kernels of a collection of related applications such as applications from the image processing domain. However, they may be capable of doing much more.

CGRAs are composed of relatively high level computational elements such as Arithmetic and Logic Units (ALUs), which are capable of doing operations such as addition, subtraction, and a simple interconnect such as communication between nearest neighbors. Figure 1 shows an example of an 8Way interconnect. 8Way is a mesh architecture where nodes can connect to any of their 8 neighbors.

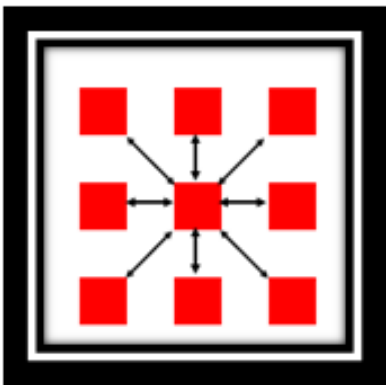


***Figure 1.** An 8Way interconnect.*

Within CGRA architectures, there is a vast design space that can be explored. Design choices include types of interconnect patterns, specialized / customized computational units, dedicated routes to pass information, and different options for routing values on and off chip. Good tools are needed to allow designers to identify efficient architectures for an application domain. Some of these design space exploration tools have already been developed, laying groundwork for progress in this direction (e.g., (Hartenstein et al., 2000b), (Karuri et al., 2008), (Bauer et al., 2009), (Kim et al., 2010)). However, one bottleneck for any of these tools is that it can be time consuming and difficult to properly evaluate a proposed architecture, especially if that architecture is quite novel or highly customized. A contributing factor is the lack of good, reliable algorithms to map applications onto new architectures, i.e., to solve the "mapping problem" in an efficient and general purpose way.

For any proposed architecture to meet the needs of an application designer, it must be possible to map desired applications onto that architecture in a way that meets the application designer's needs, whether it be fast performance, low energy consumption, small size, or typically a mix of all three.

The mapping problem is very challenging and is the focus of this manuscript. The difficulty of the mapping problem has been well discussed in the literature, and most non-trivial formulations are NP-complete (e.g., see (Garey and Johnson, 1983)). Many promising solutions have been proposed, but in practice, simulated annealing still appears as the approach of choice despite drawbacks in computation time (Bian et al., 2010). Fast, reliable alternatives are badly needed.

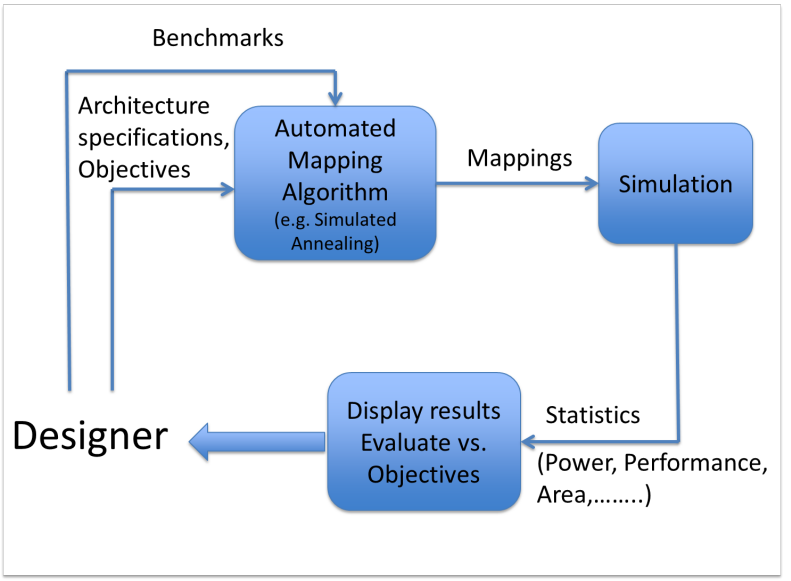
In this paper, we investigate how crowdsourcing may be applied to the mapping problem. To our knowledge, there has not been any previous research proposing to use crowdsourcing for mapping within the process of an architecture design space exploration. The potential of such an approach is shown in Figure 2. The top figure represents standard design space exploration. Here, the designer configures a specific architecture, supplies a set of benchmarks and specifies objectives (e.g., a relative importance of power vs. performance vs. area). The benchmarks are mapped onto the proposed architecture, and the resulting mapped designs are simulated. Results are fed back to the designer, who then decides how to improve the design. The role of the designer could also be replaced by an automatic algorithm that iteratively attempts to optimize the architecture against a specific objective or cost function.

Figure 2(b) shows our proposed variation on this flow. Here, we have crowd-sourced the mapping problem, relying on human flexibility, creativity, intuition, and persistence, rather than relying only on a specific, potentially limited automatic mapping algorithm. People excel at identifying opportunities and patterns and in customizing their strategy to the areas of difficulty (i.e., "pinch points") of each architecture. Furthermore, as shown by the extra arrows, we provide mechanisms for the designer to receive additional feedback in the form of informal comments from game players (see Section 6.2.1), and statistics on time to solution (how much time players took to generate mapping solutions) (see Section 6.2.2) and number of good solutions that could be collected for a given trial (see Section 6.1). This extra information can provide insights that may be valuable in selecting a final architectural design.

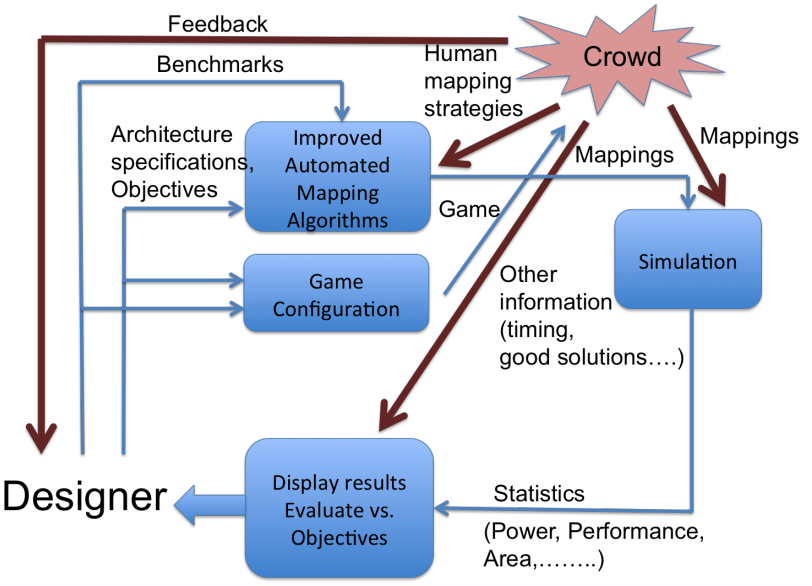
Our overall goal in this paper is to show a proof of concept of this flow, and to demonstrate that we can take advantage of the creativity of game players to provide reliable mappings and useful feedback that would be difficult to obtain in an automatic mapper. We discuss our observations as to why people perform well on this NP-complete task and also discuss opportunities to learn from crowd strategies and performance how to develop better automatic mapping algorithms. From our observations, player interviews, and the resulting database of good mappings, we have developed several new algorithms that both produce very fast results and perform 2-3 standard deviations better than Simulated Annealing (Mehta et al., 2013, 2015). For example, we have developed a data-driven algorithm based on the observation of many players that certain patterns appear in high quality mappings, and these patterns can be exploited to efficiently map new benchmarks (Mehta et al., 2013).

A portion of this research has been published in (Sistla et al., 2013). To these previously presented results, we contribute additional results, extending our initial study from five to ten architectures. Results from the five new, highly constrained architectures have not appeared in any previous publication. We also contribute additional analysis and discussion.

Specifically, in this paper, we describe results from crowdsourced mappings of 7 benchmarks to 10 architectures that amount to 8624 games in total. We find that players can outperform Simulated



(a) One standard flow of architecture exploration.



(b) Architecture exploration using crowd sourcing.

Figure 2. *(a) A standard flow where a designer uses an automated mapping algorithm to explore architectural design ideas for a given benchmark suite. Unfortunately, the ideal general purpose mapping algorithm has not yet been developed, and mapping difficulty can limit exploration. (b) We propose the use of crowds and crowd intuition to get additional insights that are hard to obtain from an automated algorithm. This qualitative feedback and human mapping strategies can be used to further improve upon the automated mapping algorithms.*

Annealing in 67 of 70 benchmark / architecture combinations. We use our results to compare architectural choices for our benchmark set and discuss our findings in view of findings for similar architectures in previous research. Results overall show good potential for including crowdsourcing as part of the design exploration process, especially for early research or exploration of basic ideas, when intuitions are needed as opposed to fine tuning. The crowd can help to identify architectures that are easy to work with and produce good solutions, constraints that are crippling, and can contribute to identification of mapping algorithm deficiencies and opportunities for better mapping algorithms.

2. RELATED RESEARCH

Our related research can be divided into four areas: coarse grained reconfigurable architectures, CGRA mapping algorithms, cross-architectural studies, and games with a purpose. We discuss each of these areas in turn.

2.1. Coarse Grained Reconfigurable Architectures

CGRAs that have been explored to date can be classified into two categories based on how functional units are arranged: stripe-based and mesh-based. Stripe-based architectures consist of one or more linear arrays of elements connected with a full or partial crossbar interconnect between rows (e.g., PipeRench / Kilocore (Goldstein et al., 2000), and RAPID (Ebeling et al., 1996)). In mesh-based architectures, functional units are arranged in a two-dimensional array having horizontal and vertical interconnect and can support nearest-neighbor, nearest-neighbor with hops, and hierarchical connections. Some of the mesh-based architectures that have been studied include MATRIX (Mirsky and Dehon, 1996), GARP (Hauser and Wawrzynek, 1997), Morphosys (Singh et al., 2000), REMARC (Miyamori and Olukotun, 1998), KressArray (Hartenstein et al., 2000a), RAW (Taylor et al., 2004), RSPA (Kim et al., 2005), ADRES (Bouwens et al., 2007), MORA (Lanuzza et al., 2007), the CGRA architecture of Kim and his colleagues (Kim and Mahapatra, 2008), and SmartCell (Liang and Huang, 2009). Research in CGRA architectural design is ongoing, and no single design has emerged as dominant. We believe that stripe-based architectures, relatively little studied, may be worth increased research investment, and we include examples of both stripe and mesh based architectures in our case studies.

2.2. Cross-Architectural Studies

A large number of cross-architectural studies have been performed for CGRAs. For example, Bansal and his colleagues (Bansal et al., 2004) and da Silva and colleagues (Silva et al., 2006) perform cross architectural studies, including investigation of alternative interconnect topologies for mesh CGRAs. Lambrechts and colleagues (Lambrechts et al., 2008; Mei et al., 2005) and Bouwens et al. (Bouwens et al., 2007, 2008) test various architectural options for ADRES like CGRAs and introduce optimizations to this architecture. Kim et al. (Kim et al., 2010) perform an architectural search over arrangements of components in non-traditional patterns to optimize design criteria. We have not seen any cross-architectural studies that compare both mesh and stripe-based architectures.

Many researchers have also worked on tools to support architectural explorations such as fast, accurate simulators and high level languages for architecture specification (e.g., (Hartenstein et al.,

2000b; Karuri et al., 2008; Bauer et al., 2009)). This research is complementary to our own and can contribute to better tools for architectural exploration.

2.3. **Mapping algorithms**

The difficulty of the mapping problem has been well discussed in the literature, and most non-trivial formulations are NP-complete (Garey and Johnson, 1983). Most existing algorithms fall into one of several styles, including greedy algorithms, randomized algorithms, clustering algorithms, Integer Linear Programming, and Analytical Placement. Comprehensive discussion and further references can be found in the following surveys: (Tehre and Kshirsagar, 2012; Choi, 2011; Theodoridis et al., 2008; Hartenstein, 2001a,b). We provide a brief summary here.

Many greedy algorithms have been explored, including deterministic place and route (Gehring and Ludwig, 1998), heuristic depth first placement (Ferreira et al., 2007), and priority order placement with backtracking (Dimitroulakos et al., 2009). Greedy or heuristic mapping is the option of choice for many mapping problems due to its speed and determinism, but for difficult problems it may perform poorly.

Randomized algorithms are pervasive, successful, and popular, due to their generality and ease of implementation. Simulated Annealing is frequently used for placement, *c.f.* VPR (Betz and Rose, 1997), SPR (Friedman et al., 2009), RaPiD(Ebeling et al., 1996), Dragon (Taghavi et al., 2005), and TimberWolf (Sechen and Sangiovanni-Vincentelli, 1985). The main drawbacks of most randomized algorithms are computation time and indeterminism.

Integer linear programming (ILP) has received attention due to its clean representation and the possibility of obtaining an optimal solution. ILP has not been shown to be feasible for large scale mapping problems (e.g., (Lee et al., 2010; Yoon et al., 2009), but it is frequently used as a component of mapping algorithms.

Partitioning or clustering algorithms promise scalability and efficiency even in large graphs, *c.f.* SPKM (Yoon et al., 2009), Capo (Adya et al., 2004), FengShui (Agnihotri et al., 2005), and NTUplace (Chen et al., 2005, 2008). Such algorithms can be very fast. However, solution quality may be suboptimal compared to algorithms that do not use divide and conquer (Sankar and Rose, 1999).

Analytical placers are efficient and scalable, and can achieve good quality results (Agnihotri and Madden, 2007; Luo and Pan, 2008). Analytical placers typically work in a two step process. A straightforward optimization is solved while allowing placements to overlap. Then, an iterative procedure is used to separate overlapping elements. As with hierarchical clustering, solution quality can suffer due to the local nature of final adjustments. Bian et al. (Bian et al., 2010) investigated the use of the FastPlace analytical placement algorithm (Viswanathan and Chu, 2005) for benchmarks having 100's of thousands of LUTs and conclude that "simulated annealing based placement would still be in dominant use for a few more device generations." As such, we use simulated annealing as the basis for comparison.

We note that other researchers have also suggested that the difficulty of mapping may be one bottleneck to widespread CGRA adoption (Cong, 2011). There is much room for improvement in any or all of these mapping algorithms, especially when we wish to allow designer flexibility to pursue out-of-the-box architecture designs. We propose substituting or supplementing traditional mappers

with input from the crowd. Results from crowdsourcing can help to alleviate mapping difficulties for the designer in initial phases of exploration, as the crowd may be able to work with architectural designs with which a mapping algorithm has difficulty, and they can also highlight difficulty or ease of mapping to specific architectures at all phases of design exploration, as crowd feedback in terms of mapping time and crowd comments can give indicators of expected mapping complexity.

2.4. Games with a Purpose

The idea of transforming computational tasks into more entertaining activities dates to at least the 1980s (Webster, 1988; Malone, 1980). Recently however, scientific games, also called Games with a Purpose (GWAP) have experienced an explosion in popularity as more researchers have realized that they can harness the knowledge of ordinary citizens in many of their problems (Terry et al., 2009; Von Ahn and Dabbish, 2004; Von Ahn et al., 2004, 2006a,b; Von Ahn and Dabbish, 2008; Cooper et al., 2010a). More information about how crowds solve problems can be found in the following books (Sunstein, 2006), (Surowiecki, 2005), and (Michelucci, 2013). Mao and his colleagues (Mao et al., 2013) have studied the performance of volunteers and paid workers on a challenging crowdsourcing task from real-world. Design for human computation is becoming a larger area of study, with a number of researchers developing taxonomies to classify efforts in this area (e.g., (Quinn and Bederson, 2011; Geiger et al., 2011)). Our game environment, UNTANGLED (Mehta et al., 2013) is most in the style of FoldIt (Cooper et al., 2010a,b), which tasks players with solving 3D protein folding problems. By placing the puzzle in the hands of thousands of players, the game's creators are able to analyze how many different players solve these puzzles and integrate their solutions into existing algorithms.

The current paper makes use of the UNTANGLED game to collect examples of mappings of various benchmarks onto various architectures. The contribution of the current paper is to demonstrate that crowdsourcing can be a viable part of the process flow in design space exploration – specifically in comparing different potential architecture designs.

Other research in crowdsourcing for EDA includes the insightful publications of DeOrio and Bertacco (DeOrio and Bertacco, 2009; Bertacco, 2012), and the game of Terry et al. (Terry et al., 2009). The potential of human computation for Electronic Design Automation (EDA) has been nicely described in the research of DeOrio and Bertacco (DeOrio and Bertacco, 2009; Bertacco, 2012), who mention, for example, the resemblance of the 1980's video game Pipe Dream (Pipe Mania (The Assembly Line, 1989)) to the problem of routing and the potential for casting placement as a packing puzzle in the manner of Tetris. As an initial step towards crowdsourcing in EDA, DeOrio and Bertacco contribute a visual game environment for solving instances of the SAT problem (Krzemien et al., 2011), specifically exploring the challenge of scalability. As a second example, Terry et al. present the game Plummings (Terry et al., 2009), where players manipulate node clusters to reduce critical path in FPGA placement. To our knowledge, there has not been any previous research proposing to use crowdsourcing for mapping within the process of an architecture design space exploration.

3. MAPPING OF APPLICATIONS ONTO A RECONFIGURABLE ARCHITECTURE

A mapping of a data flow graph (DFG) onto a reconfigurable architecture consists of an assignment of operators in the DFG to arithmetic and logic units (ALUs) in the reconfigurable architecture

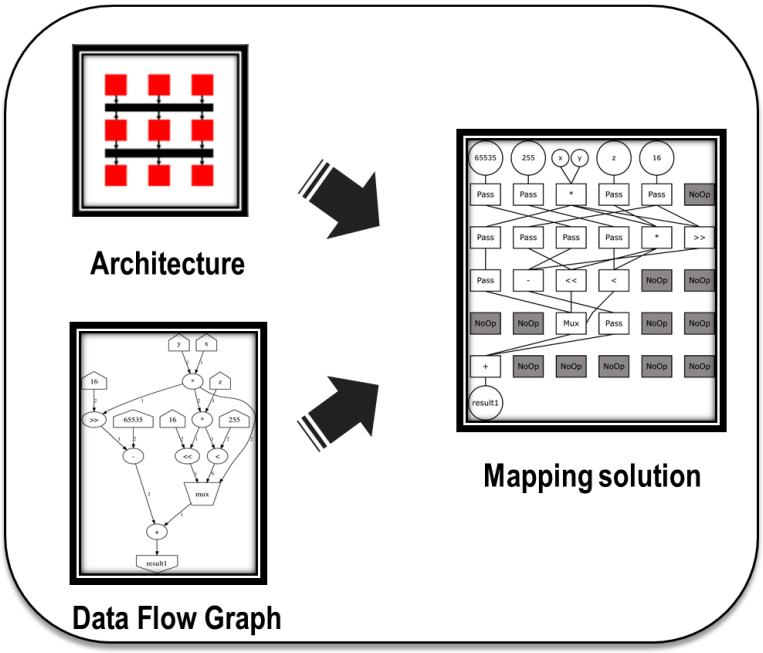
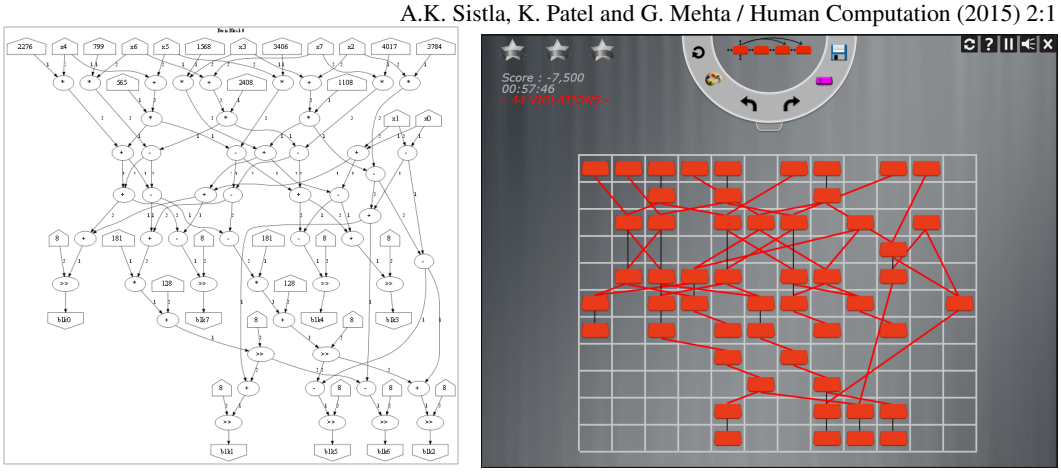


Figure 3. ***Mapping of a data flow graph onto a stripe-based architecture.***

such that the logical structure of the application is preserved and the architectural constraints of the architecture are followed. A DFG can be represented as a collection of nodes connected by edges. Edges are directional, representing flow of data from parent to child. Figure 3 shows an example of mapping a DFG of an application onto a stripe-based architecture. In the stripe-based architecture shown in Figure 3, nodes are arranged in horizontal stripes, each of which is connected to all nodes in the stripe below them using a full crossbar interconnect. Results of mapping the indicated DFG onto the stripe based architecture are shown on the right side of the figure.

4. PRESENTING MAPPING AS A GAME

Key to the success of a work flow that uses crowdsourcing for mapping is a software tool that makes the mapping problem accessible to anyone. For this purpose, we use our existing mapping game: UNTANGLED (Mehta et al., 2013). Our game interface has evolved over time based on extensive player feedback. As discussed in references on good game design (e.g., (Ambinder, 2009; Fullerton, 2008; Kennerly, 2003)), obtaining player feedback at all stages is critical. We performed numerous think aloud experiments (Ramey et al., 2006), conducted exit interviews, and encouraged players to give us feedback through emails. We have used the iterative approach of designing, testing and evaluating our game repeatedly. After we released our game online, we responded immediately to feedback from players and incorporated many of their suggestions in later iterations. To educate beginning players, we created in-depth tutorials that provide hands-on experience. To motivate players, we arranged benchmarks in order of complexity and staged tutorials to introduce concepts at each level. Incentives such as medals are given to encourage optimizing scores at individual levels.



(a) A data flow graph (H1). (b) The same graph shown on the game screen (H1).

Figure 4. ***The mapping problem consists of placing a DFG onto a given architecture. The figure on the left shows one of our example DFG's (H1), and the figure on the right shows the same graph shown on the game grid. This example maps the DFG onto a 4Way2Hops mesh architecture.***

We gave gift card incentives to winners of our online game competitions. The website for our game includes a leaderboard where players can track their scores in comparison to other players around the world. Our mapping game has received the People's Choice Award in the Games and Apps category of the 2012 International Science and Engineering Visualization Challenge conducted by the National Science Foundation and Science ((NSF, 2013), (Science, 2013)). From the designer's perspective, UNTANGLED supports stripe and mesh architectures, and it allows customization of interconnect arrangements and placement of dedicated vertical routes and input/output (I/O) blocks.

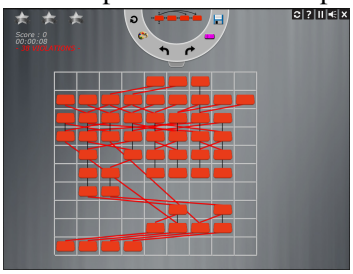
From the player's perspective, UNTANGLED presents a succession of problems where a data flow graph (DFG) must be mapped onto a specific architecture. To construct winning mappings, players optimize their scores by moving nodes of the DFG within a game grid to create compact arrangements where parents are close to their children, where "close" is defined based on the interconnect structure of the architecture.

Figure 4(a) shows an example of a DFG used in the game. This is the H1 benchmark, which is the inverse discrete cosine transform of MPEG II video compression. Here, nodes are arithmetic operators, logical operators, multiplexers, or passthroughs. The DFG is presented to players in an abstract manner, with ALU's as red rectangles shown in Figure 4(b). This representation was adequate for our examples.

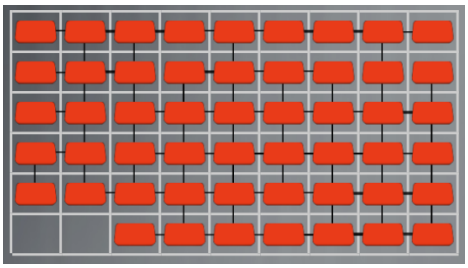
The game grid is an array of locations displayed to our players, within which nodes of a DFG can be placed. We can think of the game grid as an array of ALUs or other functional elements which have not yet been assigned operations. Placing a node from the DFG in a grid location corresponds to assigning an operation to a functional element. The architecture's interconnect defines the paths along which data can flow efficiently within the game grid. In game play, players are presented with an architecture definition, which is abstracted as an icon showing legal connections between a node and its neighbors as shown in Figure 4(b) at the top, center of the game screen.

The game interface allows players to drag and drop nodes within the game grid. Players can select and move clusters of nodes. The interface allows players to rotate and mirror their clusters. Players can add and remove passthroughs, which are often required to route data from producer nodes to consumer nodes within constraints of the architecture. Throughout the game play, players can see their score and the violations in their graph. They get incentives such as badges, medals during the game play and they can track their scores / rankings in comparison to other players around the world on the game leaderboard. The game has been continuously online since May 2012 and can be played from the website https://untangled.unt.edu/.

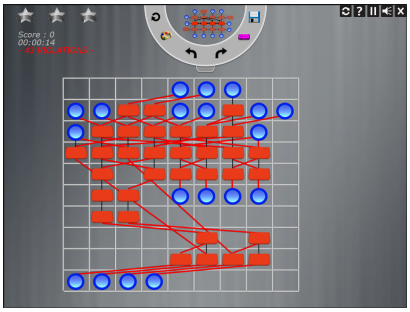
Figure 5 shows two examples of game play. Figure 5(a) shows the initial mapping of H1 from Figure 4(a) onto 4Way2Hops mesh architecture. 4Way2Hops is a mesh architecture allowing connectivity to direct horizontal and vertical neighbors, as well as horizontal and vertical connections that skip two nodes. Figure 5(b) shows a final result for one of the players. Figures 5(c) and 5(d) are initial and final graphs, mapping H1 from Figure 4(a) onto the 4Way2Hops I/O architecture. I/O blocks are shown as blue circles. This architecture is similar to 4Way2Hops with the restriction of placing I/O blocks only on the perimeter of the architecture. Note that the result in Figure 5(b) is more compact than that of Figure 5(d) in terms of grid size, but it would require more interconnect to route inputs and outputs on and off chip.



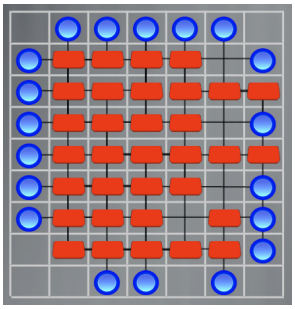
(a) Initial graph (H1).



(b) Final graph (H1).



(c) Initial graph I/O (H1).



(d) Final graph I/O (H1).

Figure 5. ***This example shows initial and final graphs of the DFG shown in Figure4(a) onto 4Way2Hops and 4Way2Hops I/O mesh architectures.***

Evaluating a mapping requires a choice of cost function. We chose a straightforward score function for players to maximize. The score of a mapping should balance power, area, and performance. Exact numbers are difficult and time consuming to compute, however, and the tradeoff between

them is application dependent. In lieu of exact numbers, we chose a proxy cost function C, designed to guide users towards a high quality mapping:

For regular architectures,

$$\begin{aligned} C &= \sum_{i=1}^{n_n}(\sum_{j=1}^{n_p} I_{i,j}) + (N_{op} * 2000) + (N_{pg} * 800) + \\ &\quad (N_{nop} * 400) + (N_{dpg} * 200) + (N_{dnop} * 40) \end{aligned} \tag{1}$$

For I/O architectures,

$$\begin{aligned} C &= \sum_{i=1}^{n_n}(\sum_{j=1}^{n_p} I_{i,j}) + (N_{op} * 2000) + (N_{pg} * 800) + \\ &\quad (N_{nop} * 400) + (N_{dpg} * 200) + (N_{dnop} * 40) + \\ &\quad (N_{IOviolations} * 300) \end{aligned} \tag{2}$$

where n_n is the number of nodes, n_p is the number of parents of a node, and parameters N are counts of elements of various types. From left to right, the cost function includes an interconnect cost $I_{i,j}$ for all edges of the mapping, a fixed cost per ALU performing an operation (N_{op}), lesser costs for ALU's used as passgates (N_{pg}) and for empty ALU's (N_{nop}), still lesser costs for dedicated route elements in the StripeDR architecture used as passgates (N_{dpg}) and performing no operation (N_{dnop}), and a cost for placing nodes requiring I/O operations in the interior of an I/O architecture ($N_{IOviolations}$). Cost factors used in Equation (1) and Equation (2) were obtained based on power simulations run on a 90 nm ASIC process from Synopsys using Synopsys PrimeTime-PX tool, but their proportions should be roughly similar for other technologies. If better numbers are known, they can be substituted trivially.

Area is taken into account as follows. At any stage of the game, we find the minimal rectangle containing all nodes as placed by the player. No-ops (empty nodes) within this rectangle are penalized as given in Equation (1). No-ops outside this rectangle are considered to be free of cost.

5. CROSS-ARCHITECTURAL STUDY: METHODS

Our objective in this paper is to evaluate the feasibility and potential benefit of performing a cross-architectural study using crowd sourcing. We have the following hypotheses: (1) People (the crowd) can solve the mapping problem in a consistent and reliable manner for a variety of architecture design choices; and (2) Crowdsourcing can be a source of additional information that is informative to a designer. This section describes the methods that we used to investigate these hypotheses.

5.1. Hypothesis 1: Consistent Mappings

To test the first hypothesis, we compare crowd mappings over a range of benchmarks and algorithms to those obtained from a general, reliable, and commonly used mapping algorithm (Simulated Annealing). We also test the ability of game players to improve upon results obtained using Simulated Annealing. We explore ten architectures within the stripe and mesh-based families, and seven benchmarks, and motivate players through two online competitions.

5.1.1. Architectures

We choose two groups of architectures for our tests, as shown in Figures 6 and 7. The first group is drawn from the most commonly studied CGRA architectures. The second group was designed to be more highly constrained, and specifically pursues the goal of reducing costly interconnect (LC variations) and costly busses needed to route inputs and outputs on and off chip (I/O variations). All architectures are described in more detail below.

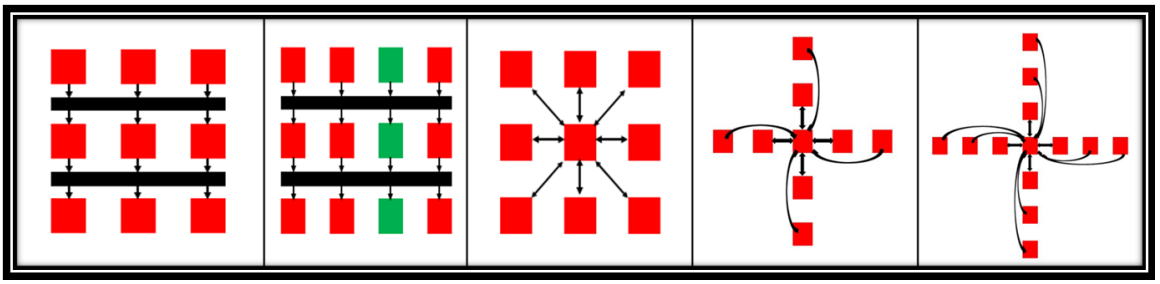


Figure 6. *Interconnect patterns for the less constrained architectures. Left to right: Stripe; StripeDR; 8Way; 4Way1Hop; 4Way2Hops.*

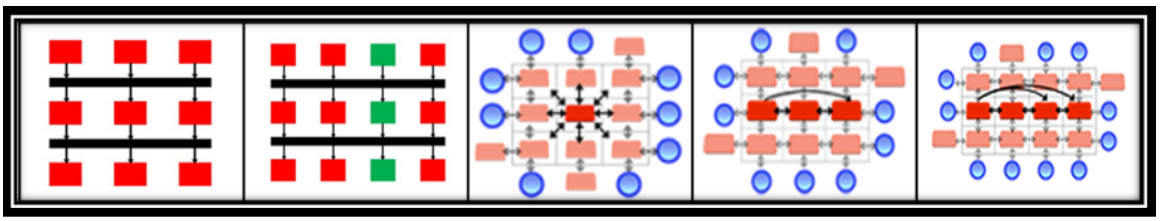


Figure 7. *Interconnect patterns for the more constrained architectures. Left to right: Stripe LC; StripeDR LC; 8Way I/O; 4Way1Hop I/O; 4Way2Hops I/O.*

The less constrained architectures are shown in Figure 6. The architectures are Stripe, StripeDR, 8Way, 4Way1Hop, and 4Way2Hop. The Stripe architecture consists of nodes arranged in horizontal stripes, each of which is connected to all nodes in the stripe below them using a full crossbar interconnect. We assume that inputs can be supplied to any ALU, but outputs (shown as green ovals) must be routed off chip, blocking all nodes below them from performing computation. The StripeDR architecture is Stripe with dedicated routes (DR) within which only passthrough nodes can be placed.

Table 1. ***Basic information related to the benchmarks.***

Benchmarks	E1	E2	E3	M1	M2	H1	H2
Nodes	24	29	29	29	36	52	61
Edges	29	29	34	36	53	63	72

8Way is a mesh architecture where nodes can connect to any of their 8 neighbors. We assume that inputs can be loaded and outputs read from any ALU. 4Way1Hop is a mesh architecture allowing connectivity to direct horizontal and vertical neighbors, as well as horizontal and vertical connections that skip one node. 4Way2Hop is a generalization of 4Way1Hop that also allows horizontal and vertical connections that skip two nodes.

Interconnect patterns for the more constrained architectures are shown in Figure 7. The architectures are Stripe LC, StripeDR LC, 8Way I/O, 4Way1Hop I/O, and 4Way2Hop I/O. Unlike Stripe which has a full crossbar interconnect, the Stripe LC architecture has a restricted interconnect where each node in a stripe is connected to only five nodes (two nodes on the left, two nodes on the right, and one node above) in the previous stripe. This level of connectivity can be achieved with an arrangement of 4:1 multiplexer interconnect (Mehta et al., 2009). The StripeDR LC architecture is Stripe with dedicated routes (DR) within which only passthrough nodes can be placed and restricted interconnect. Each node in a stripe in connected to eight nodes in the previous stripe using a straightforward 8:1 multiplexer interconnect.

8Way I/O, 4Way1Hop I/O, and 4Way2Hops I/O are similar to 8Way, 4Way1Hop, and 4Way2Hops respectively with the restriction of placing input/output blocks only on the perimeter of the chip architecture.

5.1.2. Benchmarks

We selected some of the core signal processing benchmarks from the MediaBench benchmark suite and some edge detection benchmarks from the image processing domain for our case studies. The seven benchmarks for our study included the Sobel (E1) and Laplace (E2) edge detection benchmarks, as well as GSM (E3), ADPCM decoder (M1), ADPCM encoder (M2), IDCT row (H1), and IDCT col (H2) benchmarks from the MediaBench suite. Basic statistics for the DFG's for these benchmarks are shown in Table 1. The benchmarks varied in the density and arrangement of interconnections, with the simplest graphs being easy to lay out in an efficient manner and the hardest graphs quite difficult. The four easiest graphs were planar (i.e., they could be placed in 2D with no edge crossings). The three most difficult graphs were not.

5.1.3. Obtaining Participation

We first tested the ability of players to create mappings from scratch. To jump start participation, we hosted our first worldwide online game competition in 2012, which took place for a period of ten days from August 10th through August 20th. Current players were encouraged to participate in the competition and new players were recruited using a variety of resources including online press releases and university-sanctioned posts on social networking websites. This competition tested all

benchmarks with the five architectures shown in Figure 6. To gain an influx of participation for the new architectures, shown in Figure 7, we hosted another competition in 2013 which lasted for seventeen days from September 1st to September 17th.

For both online game competitions, gift card incentives were provided to winners of each architecture and overall competition. To determine the top players, points were distributed to players based on their ranking for every architecture and graph. Ranks were determined first based on violations, with fewer violations ranking more highly. Among players with the same number of violations, rank was determined according to score. Rankings were visible on the UNTANGLED leaderboard throughout the competitions and can still be viewed on the game websites https://untangled.unt.edu/competition/home.php and https://untangled.unt.edu/competition2013/home.php. In the results reported here, we use all played games, including those contributed after the end of the competitions.

We also tested the concept of using crowd sourcing to improve upon a traditional mapping algorithm. We engaged some of our more experienced players to attempt to improve upon results obtained using our own customized Simulated Annealing algorithm. We presented these players with best, average, and poor results from Simulated Annealing runs for all benchmarks and architectures and asked them to spend no more than a few minutes on each example to attempt to improve score.

5.1.4. Our Simulated Annealing Algorithm

Simulated Annealing (SA) is frequently used for placement because of its flexibility and good quality performance. We developed our own SA algorithm which closely follows that of Betz (Betz, 1997), with the exception that we do not use SA for placement only, but perform routing in the inner loop of the algorithm. After any pair of nodes are swapped in the inner loop of the algorithm, the entire graph is routed, and the score is calculated based on that routing. To route the graph, we use a flood fill algorithm to find a legal route for each "long edge" that is not directly supported by the architecture. The flood fill algorithm explores all possible routes in a breadth-first manner from start and goal simultaneously, returning the first path where the start and goal search trees meet. This path is filled with passgates to connect the parent and child, and the algorithm continues until the flood fill algorithm fails or the complete graph has been routed. The same cost function (given in Equation (1)) has been used in SA and the game.

Because routing in the inner loop may complicate the search landscape for the SA algorithm, we ran convergence tests by repeatedly doubling the number of moves allowed at each iteration. We found that multiplying the number of moves recommended in (Betz, 1997) by a factor of 4 allowed for convergence of the SA algorithm even for our most difficult benchmark / architecture combination. Based on our findings, for our experiments, we first calculate the number of moves at each iteration as specified by (Betz, 1997) and then multiply that value by 4 before proceeding. As a result, the algorithm takes longer to run, but we can be more certain of having good results to compare to those of the players. We also note that when we do not route in the inner loop in the manner that works well for VPR (Betz, 1997), results of SA for the architectures considered in this manuscript are much worse; many placements are selected for which no routing solutions are available due to the highly restricted interconnect. Due to this reason, other SA mappers for CGRAs have also used a similar approach, with routing in the inner loop (Sharma et al., 2005). Simulated Annealing uses random numbers to determine which swaps to execute, and so results do vary from run to run. We

Table 2. ***Number of Games Played - Original Architectures - Competition 2012***

	Number of Players	Stripe	StripeDR	8Way	4Way1Hop	4Way2Hops	Total
Competition 2012	161	686	256	193	125	208	1468
Overall	824	2908	1209	1099	462	703	6381

Table 3. ***Number of Games Played - New Architectures - Competition 2013***

	Number of Players	Stripe-LC	DR-LC	8WayI/O	4Way1HopI/O	4Way2HopsI/O	Total
Competition 2013	135	397	335	341	212	484	1769
Overall	216	497	412	438	245	651	2243

ran SA ten times for each architecture / benchmark combination and compare player results to the best of these runs. We also measure player performance in terms of standard deviations from the SA mean.

5.2. Hypothesis 2: Additional Information

Ultimately, we wish to streamline the process of collecting additional information about each architecture. To explore feedback about architectures, we performed several user studies. The experimental protocol for all studies was determined to qualify for an exemption from the Institutional Review Board of our university (information removed for blind review). IRB protocols were followed in all cases. We conducted our studies using the think-aloud process (Ramey et al., 2006). Players were given an introduction to the game and invited to play, while talking out loud about what they were doing and their goals and expectations at each step.

In addition, we solicited feedback from players during and after our online competition. Our qualitative results are drawn from all of these sources. For quantitative results, we look at time spent to create good mappings for each architecture.

6. RESULTS

Our 2012 competition attracted 161 players, who contributed a total of 1468 completed games (Table 2). Players contributing to the competition played an average of 4 hours over the course of ten days, with 37 players (23%) exceeding 3 hours of game play and 21 players (16%) exceeding 10 hours of game play. The competition saw a spirited battle between two top contenders for overall winner.

Our 2013 competition attracted 135 players, who contributed a total of 1769 completed games (Table 3). Players contributing to the competition played an average of 3 hours over the course of 17 days, with 31 players (24%) exceeding 3 hours of game play and 10 players (8%) exceeding 10 hours of game play.

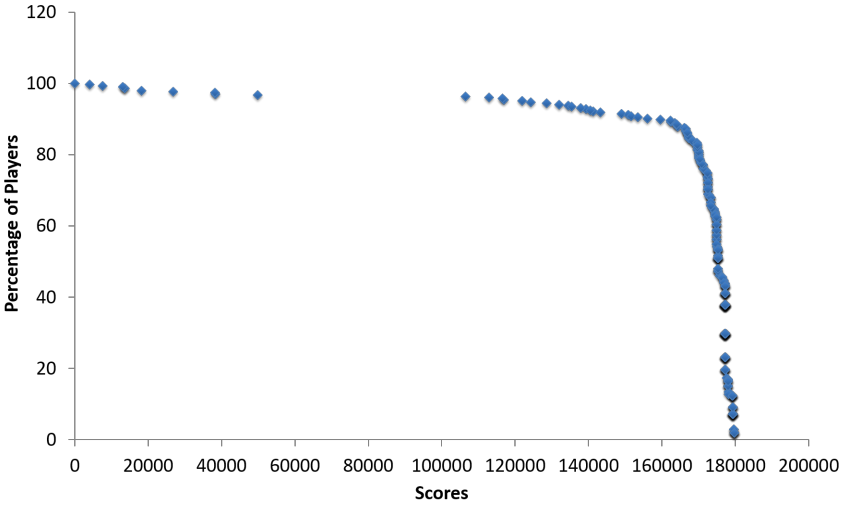


Figure 8. ***Distribution of scores of our players for 8Way architecture and E1 level.***

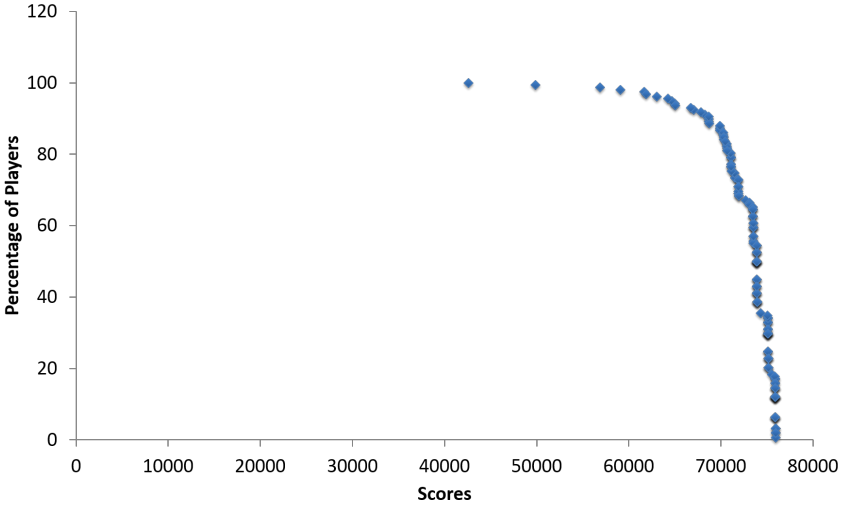


Figure 9. ***Distribution of scores of our players for 8Way architecture and M1 level.***

Outside of the competition, we continued to collect results. In total, results from 750 players and 8624 total games were used in generating player vs. SA comparisons and obtaining timing information and feedback from players. Quotes from players are uniformly positive. Examples include: “I had a blast playing your game, Untangled." “I thoroughly enjoyed playing in the Untangled competition." “I enjoyed the general idea of the game, the feel of the competition, and I also enjoyed having different kinds of puzzles, rather than just one kind."

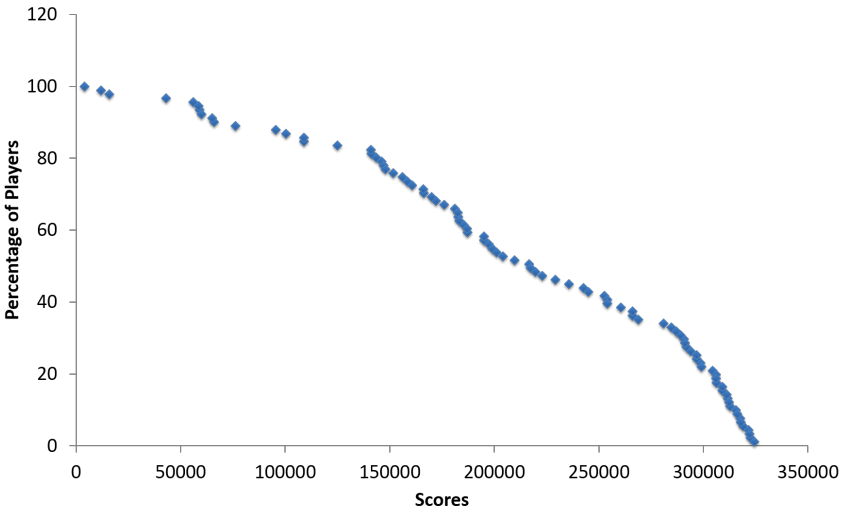


Figure 10. ***Distribution of scores of our players for 8Way architecture and H1 level.***

Figure 8, Figure 9, and Figure 10 show scores of players on x-axis and percent of players who achieved that score or higher on the y-axis for our 8 Way architecture for E1, M1, and H1 levels respectively. We chose an 8 Way architecture as example to show the trend for easy, medium and hard difficulty levels. We observed a similar trend in other architectures as well.

6.1. Reliability of player mappings

To explore reliability of player mappings, we compare player results to those from the SA algorithm, beginning with a comparison of best results. Table 4 and Table 5 show the percentage difference in maximum scores between the top player and the top SA run for each condition. Tables entries are calculated as a percentage difference of top ranking mappings. For example, for the "Stripe LC" architecture and the "E1" DFG, the score of the top player was 5.6% better than that of the best simulated annealing run. Instances where the player outperforms SA are highlighted in bold. Players outperformed SA for all benchmarks in Stripe, StripeDR, Stripe LC, StripeDR LC, 4Way2Hop, and 4Way2Hop I/O architectures. SA was able to find the best solution in just 5 of the 70 cases. The overwhelming success of players vs. SA is notable, because SA, due to its random nature, is completely capable of finding the globally optimal, and therefore guaranteed best solution.

A designer may not always wish to or have time to perform multiple SA runs, and so it is also useful to compare player results to the mean result returned by SA. Table 6 and Table 7 show player score advantage measured in the number of standard deviations greater than the mean of the SA runs. From Table 6, we can see that on average, the player results are 2.1 standard deviations better than the mean SA run for the less constrained architectures. Table 7 shows that on average, the player results are 3.5 standard deviations better than the mean SA run for the more constrained architecture. From these tables, we see that the best player score was greater than the SA mean in 67 of 70 cases. The exceptions were H1 on the 4Way1Hop and 8Way I/O architectures, and benchmark

Table 4. ***Percentage score advantage, players vs. SA Best of 10 Runs - Regular stripe and mesh architectures***

	Stripe	StripeDR	8Way	4Way1Hop	4Way2Hops	Average
E1	**10.4**	**5.4**	**1.1**	**1.7**	**1.2**	**3.9**
E2	**16.2**	**3.0**	**0.0**	**1.6**	**1.6**	**4.5**
E3	**2.7**	**2.0**	**0.1**	**4.0**	**4.1**	**2.6**
M1	**3.6**	**2.5**	**2.6**	**0.9**	**1.6**	**2.3**
M2	**2.0**	**1.6**	**2.6**	-2.4	**1.2**	**1.0**
H1	**5.0**	**2.5**	-1.4	-2.8	**0.8**	**0.8**
H2	**4.8**	**1.3**	**0.4**	0.0	**2.1**	**1.7**
Average	**6.4**	**2.6**	**0.8**	**0.4**	**1.8**	**2.4**

Table 5. ***Percentage score advantage, players vs. SA Best of 10 Runs - Stripe and mesh architectures with additional constraints***

	Stripe LC	DR LC	8Way I/O	4Way1Hop I/O	4Way2Hops I/O	Average
E1	**5.6**	**3.0**	**1.1**	**1.4**	**1.7**	**2.6**
E2	**4.0**	**3.4**	**2.0**	**2.3**	**2.7**	**2.9**
E3	**1.0**	**1.1**	**1.8**	**3.4**	**6.5**	**2.8**
M1	**1.7**	**1.3**	**2.6**	**5.6**	**4.7**	**3.2**
M2	**1.5**	**1.8**	**1.2**	**1.1**	**0.8**	**1.3**
H1	**3.0**	**1.7**	-2.9	-0.5	**2.8**	**0.8**
H2	**2.1**	**1.0**	**1.6**	**5.9**	**5.7**	**3.3**
Average	**2.7**	**1.9**	**1.1**	**2.8**	**3.6**	**2.4**

M2 on the 4Way1Hop architecture. In these tables, it is notable that players often performed SA by a substantial margin in terms of standard deviations – often by two standard deviations or more.

6.2. Additional Information

Having shown that players can provide reliable mappings, we now discuss additional information obtained from the players.

6.2.1. Strategies

One type of information we can obtain from players concerns strategies that they use to create high quality mappings. We can observe players strategies in the accompanying video. Examples from game play are also given in Figure 11, Figure 12, Figure 13, and Figure 14. All of these examples illustrate player ability to capitalize on observed opportunities. In Figure 11, we see a typical move to compact a graph by reducing the number of columns. Players readily spot interior blank spaces and quickly learn how to slide and pivot nodes to fill those spaces and compact the graph. Figure 12 shows a more complex example of this type of rearrangement, which suggests a type of search

Table 6. ***Score advantage, players vs. SA mean of 10 Runs, measured in number of standard deviations - Regular architectures***

	Stripe	StripeDR	8Way	4Way1Hop	4Way2Hops	Average
E1	**2.8**	**2.0**	**2.4**	**5.3**	**2.9**	**3.1**
E2	**3.1**	**2.0**	**1.3**	**2.7**	**2.4**	**2.3**
E3	**2.1**	**1.8**	**2.6**	**5.5**	**5.0**	**3.4**
M1	**1.8**	**2.0**	**2.1**	**2.2**	**2.2**	**2.1**
M2	**2.0**	**1.5**	**2.4**	-0.8	**2.8**	**1.6**
H1	**3.4**	**2.7**	**0.1**	-1.8	**3.1**	**1.5**
H2	**4.1**	**6.3**	**1.6**	**1.5**	**3.2**	**3.3**
Average	**2.7**	**2.6**	**1.8**	**2.1**	**3.1**	**2.5**

Table 7. ***Score advantage, players vs. SA mean of 10 Runs, measured in number of standard deviations - Architectures with additional constraints***

	Stripe LC	DR LC	8Way I/O	4Way1Hop I/O	4Way2Hops I/O	Average
E1	**2.9**	**1.6**	**3.4**	**3.1**	**5.3**	**3.3**
E2	**5.3**	**2.0**	**10.0**	**4.4**	**4.4**	**5.2**
E3	**2.4**	**1.7**	**2.1**	**2.8**	**9.6**	**3.7**
M1	**1.8**	**1.9**	**2.0**	**2.9**	**4.6**	**2.6**
M2	**3.5**	**2.6**	**2.0**	**2.1**	**1.9**	**2.4**
H1	**5.0**	**2.0**	-0.2	**2.2**	**4.8**	**2.8**
H2	**5.2**	**5.0**	**2.3**	**5.2**	**6.0**	**4.8**
Average	**3.7**	**2.4**	**3.1**	**3.2**	**5.2**	**3.5**

process to identify a set of sliding and pivoting moves that can fill the interior spaces and reduce the number of rows and columns.

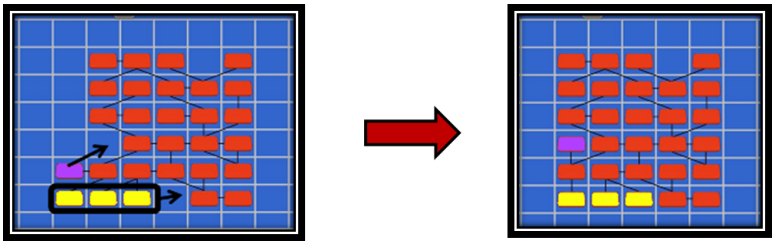


Figure 11. *Solving 8Way, E2: pivoting. A player uses two coordinated moves to improve upon a graph generated by SA.*

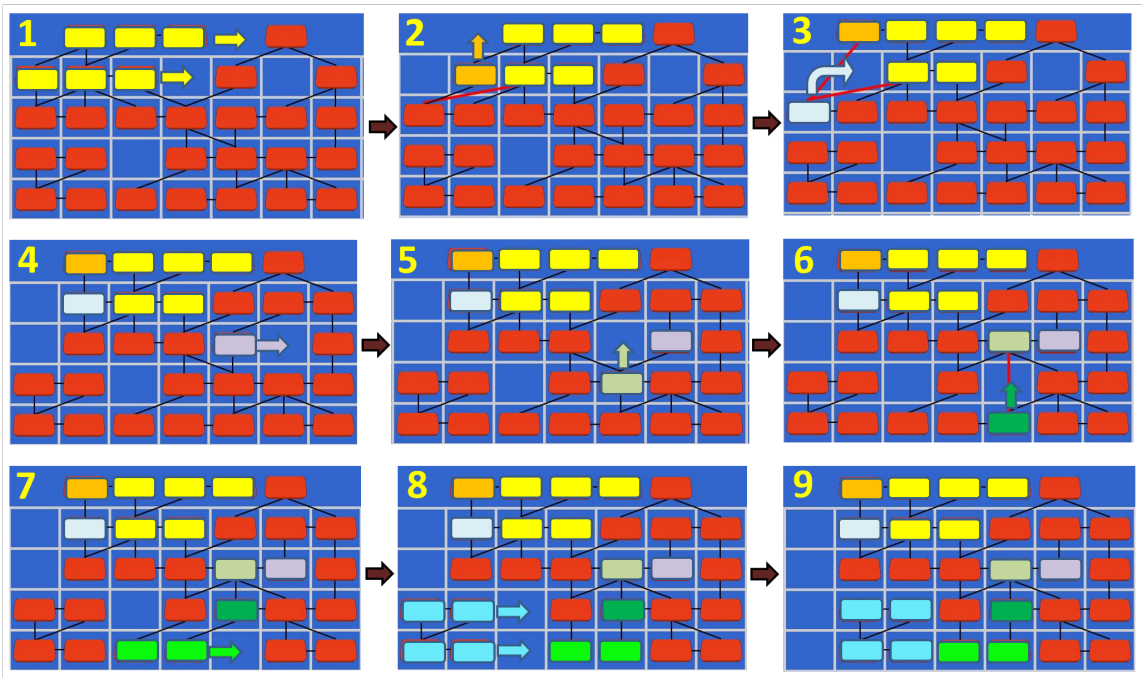


Figure 12. *Solving 8Way, M1: pivoting. A player uses a sequence of coordinated moves to improve upon a graph generated by SA.*

Figure 13 shows an example for the StripeDR architecture. Here, a player reduces the number of columns by scheduling a pair of operations to happen later in the execution path, so that the right hand column can be removed. Figure 14 shows an example with the 8Way I/O architecture. Here a complex series of group rotation, sliding, and pivoting steps is used to compact the graph while maintaining the I/O nodes on the edges of the final graph.

Strategy information can also be obtained from introspective comments of the top game players.

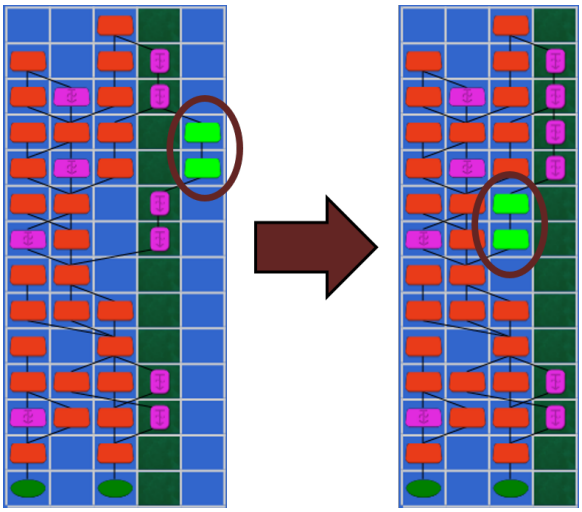


Figure 13. *Solving StripeDR, M1: passgate rearrangement. (Left) The right hand column of this graph contains two nodes that cannot be moved into the DR column. (Right) Scheduling these nodes later in the computation reduces the overall width of the graph.*

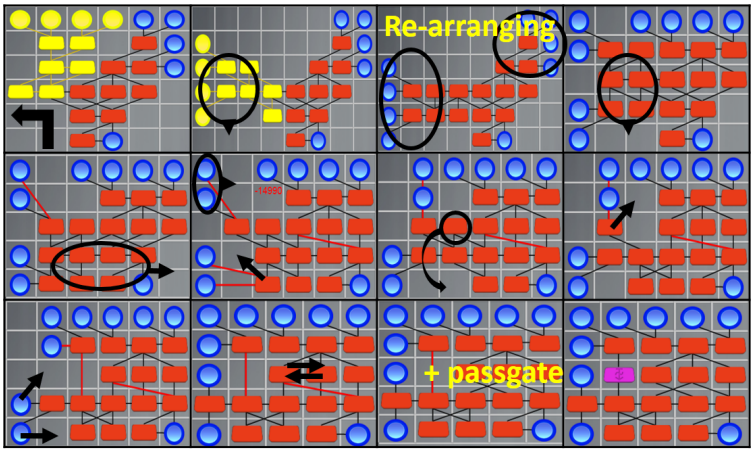


Figure 14. *Solving 8Way, E1: A player uses a sequence of coordinated moves to optimize the graph to reduce the width and height of the fabric.*

Here are some examples: for stripe and DR architectures, "I began working on complex graphs by focusing on the highest-degree nodes, and in Stripe and DR I usually accompany this strategy with a top-to-bottom strategy, solving violations near the top of the graph before spending time on the ones at the bottom of the graph. I also in some cases was able to reduce the total surface area of the graph by adding passgates to increase the height of some chains of nodes near the top of the

Table 8. Player Mapping Times (sec) - Minimum over top 5 rank players, less constrained architectures

	Stripe	StripeDR	8Way	4Way1Hop	4Way2Hops	Average
E1	427	547	280	87	121	**292**
E2	497	296	84	63	126	**213**
E3	253	381	118	106	134	**198**
M1	372	274	267	287	151	**270**
M2	907	664	543	480	359	**591**
H1	1144	582	614	762	242	**669**
H2	1298	1136	541	538	633	**829**
Average	**700**	**554**	**350**	**332**	**252**	**438**

graphs, because this often freed up some space below in the process and thus allowed me to reduce the width of the graph. ”

For regular mesh architectures, “Typically, the blocks that were initially in the center of the grid had more connections on them and would therefore have to be arranged somewhat around each other in the center of the solved puzzle while the blocks with merely one or two connections could easily be placed around the outside of the solved puzzle in the end.” My wife suggested coloring the blocks according to their degree, which seemed to help keep track of which blocks were more important in the long run (blocks with higher degrees are usually harder to place optimally).”

“For I/O games, my strategy initially focused on moving all output nodes to the location along the perimeter of the graph in which they were closest to their connecting nodes. After that, I worked on solving internal violations, moving output nodes to different locations along the perimeter (or making the perimeter/graph size larger) as needed.”

“On the 4-way I/O games, I initially prioritized rectilinearity over proximity, looked for configurations that reduced the number of diagonal connections, then look for ways to improve proximities by interchanging rows or columns. Part of this was at least somewhat guided by structures which pretty much dictate their own arrangement - sets of three mutually connected nodes must be aligned linearly, as must sets of four nodes where two nodes each connect to all three of the others but the remaining two are not connected, for instance. Identifying such structures early on also helped in getting to a rectilinear configuration in the first place.” “On the 8-way I/O, start by finding the nodes with the largest numbers of connections and trying to get those connections satisfied first.”

Motivated by the best players, we suggest placing patterns of nodes at each step, rather than placing individual nodes one by one. We observe that the experienced players are always looking beyond the next node to be placed, and they will settle quickly into particular mappings for collections of nodes connected in familiar patterns. Knowledge of arrangements of connected patterns that are low-cost or are known to be useful should be of great help in organizing a “mapping search tree” of manageable size. In the revised search tree, instead of traversing the tree by placing nodes one by one, we would instead traverse the tree by placing collections of neighboring nodes into high probability patterns.

Table 9. Player Mapping Times (sec) - Minimum over top 5 rank players, more constrained architectures

	Stripe LC	DR LC	8Way I/O	4Way1Hop I/O	4Way2Hops I/O	Average
E1	275	449	183	107	146	**232**
E2	390	293	77	122	411	**259**
E3	225	605	119	91	74	**223**
M1	356	551	189	540	287	**385**
M2	2003	795	1968	545	928	**1248**
H1	638	1034	4199	1294	1353	**1704**
H2	763	2227	1409	1867	1028	**1459**
Average	**664**	**851**	**1163**	**652**	**604**	**787**

6.2.2. Difficulty

Another type of information we can obtain from players concerns the perceived difficulty of mapping to each architecture.

When asked to rank the relative difficulty of the less constrained architectures, players gave the following partial order from easiest to most difficult: (1) 4Way2Hop and Stripe; (2) 4Way1Hop and StripeDR; (3) 8Way. If we examine the time taken to perform the mappings for regular stripe and mesh architectures (Table 8), it matches this ordering for the mesh architectures: 4Way2Hop, 4Way1Hop, then 8Way. However, the stripe architectures required significantly longer to generate a solution. We can partially explain this result based on a learning effect. Almost all players attempted Stripe and StripeDR before attempting any of the mesh architectures. These two architectures were their first exposures to the game and thus may have required additional time to learn how to use the tools and how to improve their score.

Although players perceived 4Way1Hop to be easier than 8Way and spent more time with 8Way, they performed better on 8Way, outperforming Simulated Annealing in all cases. The perceived difficulty of 8Way appeared to be that it did not provide sufficient flexibility to route graph crossings. In this regard, 4Way1Hop, despite having the same number of connected neighbors as 8Way, was perceived to be easier to manipulate. In practical terms, however, this perceived difference does not appear to prevent players from generating good graphs. One player's comments may give insight: "There's enough flexibility with 8-way that there was never much trouble fitting things in a target rectangle, so minimizing the pink blocks (only an issue on the hardest levels) and the number of crossings was the main challenge."

For the more constrained architectures, player consensus on relative difficulty was (1) Stripe LC, (2) Stripe DR LC, (3) 4Way2Hop, (4) 8Way, and (5) 4Way1Hop. However, 4Way2Hop, 8Way, and 4Way1Hop were considered to be similar in difficulty level, with the Stripe variations being much easier. Note that the added constraints of the I/O architecture pushed the mesh architectures to be much more difficult than the stripe variations, whereas they were much more similar in difficulty in the less constrained architectures. Player timing information (Table 9) only partially supports this self assessment of difficulty. In particular, players spent as much time on the Stripe variations as on the mesh variations, and they spent by far the greatest time on the 8Way architectures.

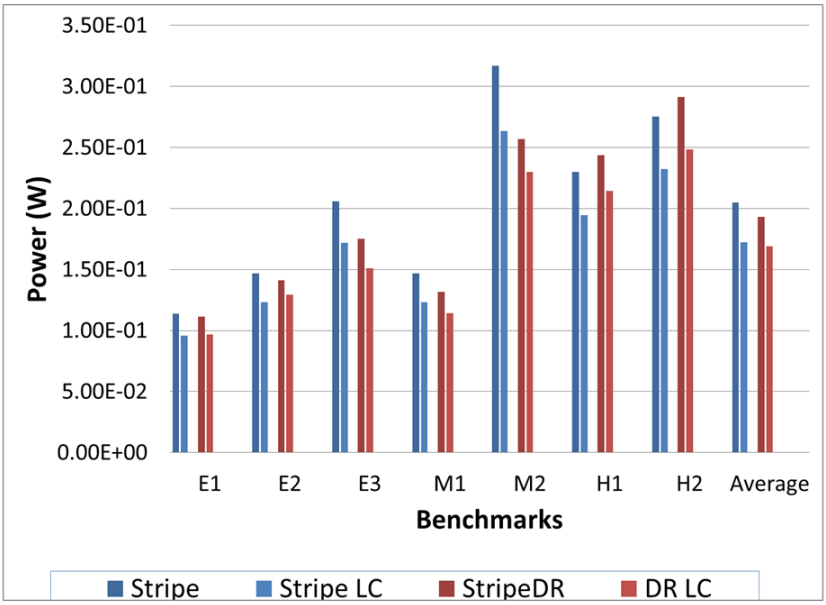


Figure 15. *Architecture comparison based on power consumption for stripe-based architectures.*

Overall, players found the more constrained architectures were very difficult to map to because of the extra constraints added to these architectures. They have to obey not only the connectivity constraints of the architecture but also the placement of I/O blocks on the perimeter of the chip architecture. That makes the mapping problem more challenging for the more restricted architectures, and this additional challenge can be seen in player solution times (i.e., compare timings in Table 8 and Table 9).

6.3. Comparison of Architectures

Finally, we compare all the architectures using the best mapping results obtained from the top scoring players for stripe and mesh architectures and benchmarks. In Figure 15, we compare stripe-based architectures. The StripeDR architecture consumes 6% less power as compared to the Stripe architecture as it has efficient dedicated vertical routes for passing data from producer computational units to consumer computational units. If we compare Stripe LC, which has limited connectivity, with the regular Stripe architecture we can see that it consumes 16% less power. The DR architecture with limited connectivity (StripeDR LC) consumes 12% less power as compared to StripeDR.

Figure 16 shows that the DR architectures require less area than their counterparts (e.g., StripeDR vs. Stripe), because the dedicated routes require less area than regular ALUs. However both the less constrained and the more constrained architectures consume approximately the same area (e.g., StripeDR vs. StripeDR LC) because top players were still targeting for very compact mapping solutions even when the limited connectivity constraint was not given to them.

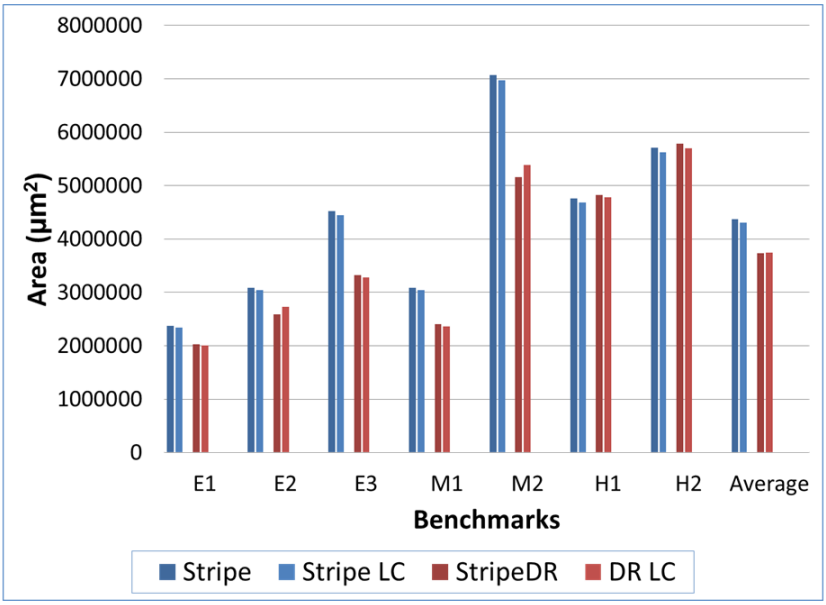


Figure 16. *Architecture comparison based on area for stripe-based architectures.*

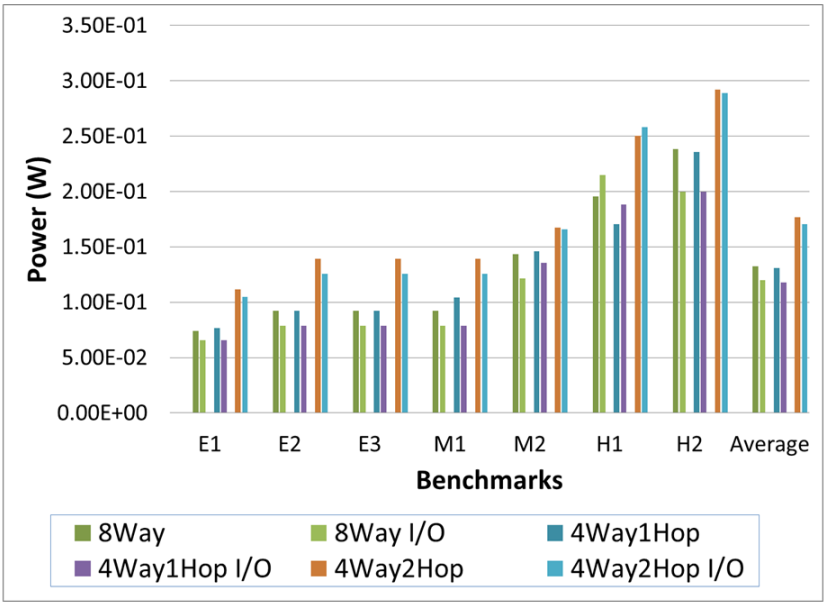


Figure 17. *Architecture comparison based on power consumption for mesh-based architectures.*

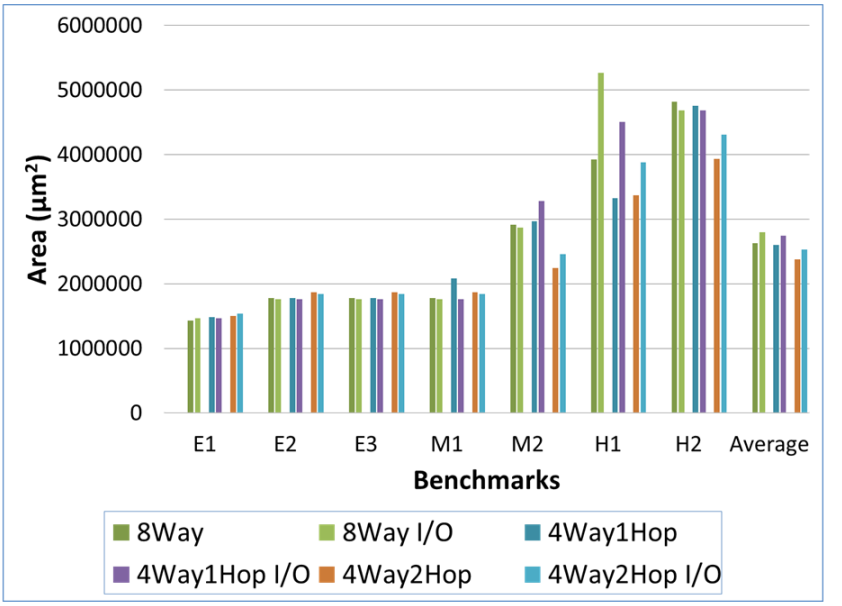


Figure 18. *Architecture comparison based on area for mesh-based architectures.*

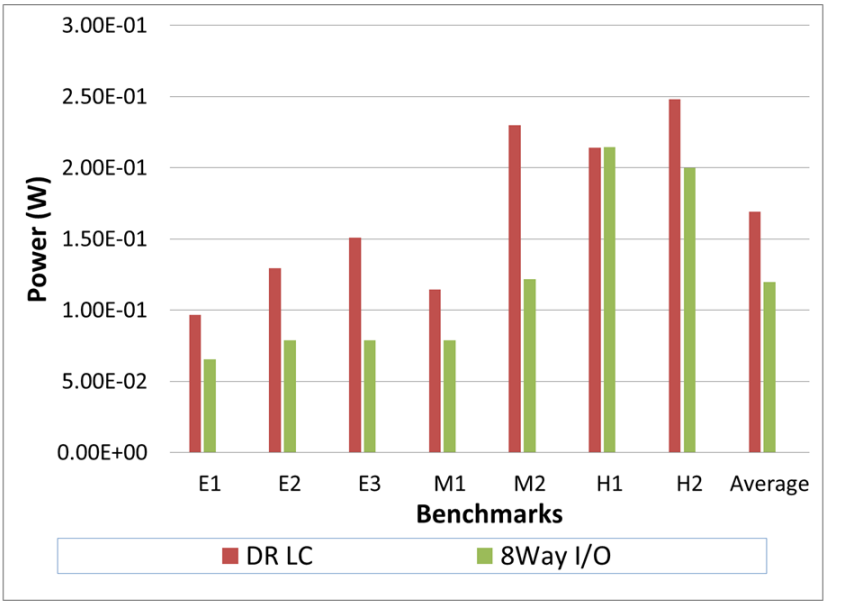


Figure 19. *Architecture comparison based on power consumption for mesh-based architectures.*

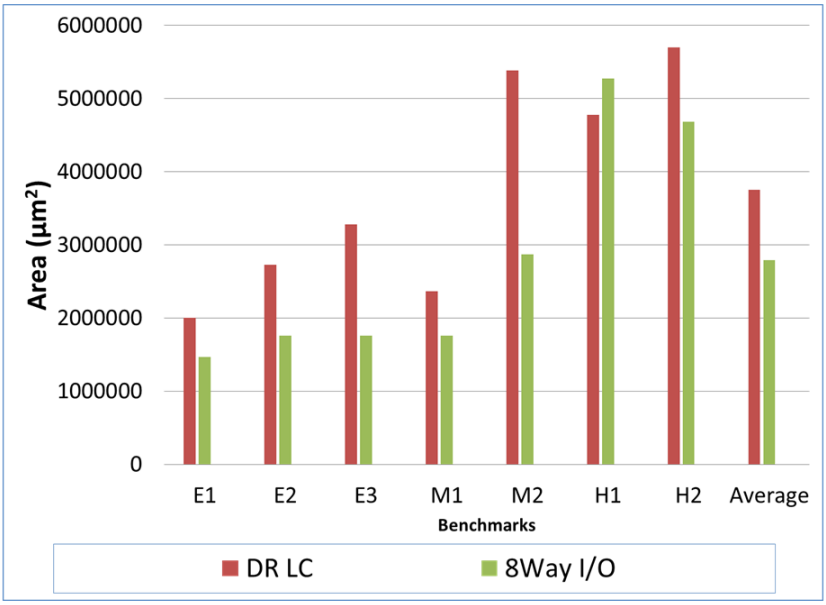


Figure 20. ***Architecture comparison based on area for mesh-based architectures.***

Based on both energy and area results, if we were to choose a stripe based architecture following this study, the winner for our benchmark set would be Stripe DR LC.

Figure 17 shows a comparison of mesh architectures based on power consumption. Values are broken out by benchmark and also presented as an average. From this figure, we can see that 4Way2Hops I/O, 4Way1Hop I/O and 8Way I/O consume less power as compared to their regular architecture counterparts. For example, 4Way1Hop I/O consumes 10% less power as compared to 4Way1Hop.

Figure 18 shows a comparison of mesh architectures based on area. The I/O architectures consume more area as compared to the regular architectures. For example, 8Way I/O consumes 6% more area as compared to the regular 8 Way. The I/O architectures require more area, because the most compact mappings cannot always be achieved while maintaining all inputs and outputs at the chip boundaries. However, the energy savings remain, and we can summarize by noting that the mesh architectures with I/O placement constraints can provide some power savings at the cost of small area overhead.

Choice of a mesh architecture based on our results would depend on the relative importance of energy vs. area. Thus, either 8Way I/O (less energy) or 8Way (less area) may be a good choice, as could 4Way1Hop I/O (less energy) or 4Way1Hop (less area). The difficulty and reliability of mapping to any of these architectures could become a dominant factor in this choice.

When we compare the top stripe based architecture – Stripe DR LC – to the top mesh based archi-

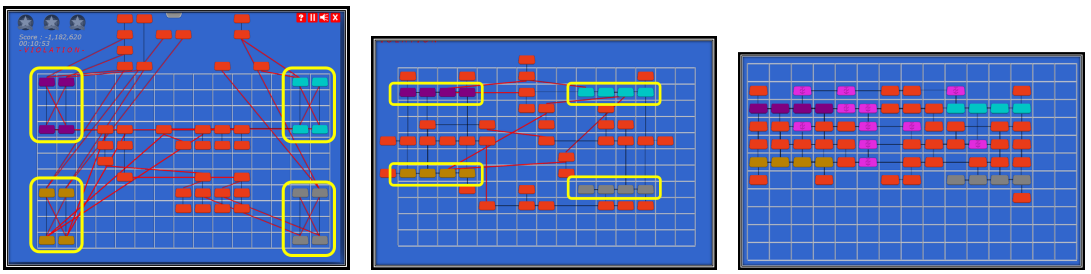
tecture, say 8Way I/O, the mesh architecture shows significantly better power and area values. As it can be seen from Figure 19 and Figure 20, 8Way I/O consumes 29% less power as compared to DR LC and it takes 25% less area as compared to DR LC. It is intriguing, however,r that the most difficult benchmark, H1, shows similar numbers for both architectures. The stripe based architectures were found to be relatively easy to map to, especially compared to the mesh I/O variants. In addition, they may lend themselves to certain optimizations in design. For more difficult benchmarks, it is possible that DR LC may be the better choice.

7. MAPPING IS NP-COMPLETE. WHY MIGHT PEOPLE BE ABLE TO SOLVE THIS PROBLEM?

In this section, we discuss the observations we made by observing and interviewing players of UNTANGLED. We see that players are very good at the following, all of which help them to solve this difficult problem effectively:

(1) **Recognizing opportunities / salience of opportunities.** Sometimes players rotate or relocate an entire group of nodes that will produce a much nicer fit of that collection of nodes to the rest of the graph. To the player, this grouping of nodes appears salient, and the opportunity to rotate or relocate for a better fit appears to pop out. Figure 14 shows one use of rotation in a longer sequence of actions used to compact a graph.

(2) **Identifying bottlenecks / key difficulties.** We also see that players, simply as a result of manipulating a puzzle and attempting to solve it are able to identify the parts of the problem that are easy and the parts that are difficult. Once the difficult part of the problem has been identified, the player can concentrate on solving that part of the problem, leaving the easy parts for later. This strategy effectively reduces problem size without really losing much in terms of ability to find an optimal (or near-optimal) solution. Figure 21 shows an example, where a player has identified, color coded, and arranged collections of nodes having a particular connectivity pattern prior to solving the remainder of the mapping.



(a) Separating clusters. (b) Solving clusters. (c) Final result.

Figure 21. ***Solving 4Way2Hop, H2. This player's strategy is to separate the graph into familiar subclusters (Left), resolve those subclusters (Middle), and bring them together to compact the graph (Right).***

(3) **Developing useful abstractions.** The pattern recognition displayed in Figure 21 is a common feature of experienced players' strategies that also helps them to abstract away parts of the problem.

After players have some experience, they begin to recognize that certain configurations of connected nodes can only be placed into an architecture in a small number of patterns. They can then iterate through those patterns to see which of them are most convenient in their solution. We have been able to exploit this idea to develop an automatic algorithm that is data-driven (based on good results from many mappings) and pattern based, working at a higher level of abstraction than individual nodes (Mehta et al., 2013).

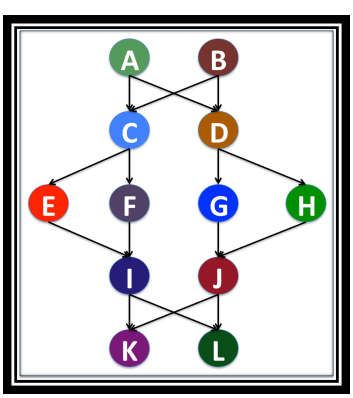
In addition, we observe two characteristics of the problem which may help to make crowd participation feasible. The problem is NP-complete, and so there are no guarantees and there may be certain worst-case examples. However, on average, many of these problems have not just one solution that is far better than others, but instead may have a large basin of solutions, all of which are pretty good, perhaps even near-optimal. For example, Figure 22 shows a histogram of solution counts for the data flow graph shown in Figure 22(a) implemented on a 4Way2Hops architecture (Figure 22(b)). There are few solutions at the very optimal end, but it should be relatively easy to find a solution within 5% of optimal, as even for this simple example, there are hundreds of thousands of possible solutions in this range.

This characteristic of a wide basin of solutions appears to vary architecture by architecture, which has strong implications for architecture design. For example, the more highly connected architectures seem to have more near-optimal solutions than the less highly connected architectures. More interestingly, the particular connectivity pattern (e.g., diagonal connections vs. only horizontal and vertical) also appears to affect the size of this basin. In situations where there is a large basin of good solutions, it can be much easier for almost any organized attempt to solve the puzzle to fall into an adequate solution. And an adequate solution may be sufficient to allow an architect to explore alternative designs, at least in an initial pass of exploration.

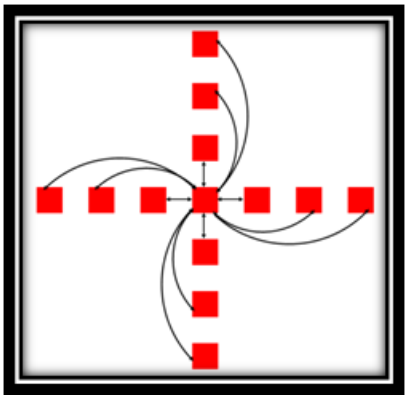
Second, there are often many ways to get to a good solution. One example is that our players have discovered an extremely effective algorithm in the 8Way architecture, where computational elements are connected to their eight closest neighbors. In this architecture, a simple algorithm that often works very well is to pivot nodes one by one about their parents or children to move nodes from the outside of a partially solved graph into the inside for a compact and efficient result (Figures 11 and 12). It is not so important which nodes are pivoted when, as long as progress is made to making the solution more compact, an outcome that is very evident to the player. A second example here are symmetries. There may be many symmetric solution choices, and it often does not matter much which solution a player will pick.

8. DISCUSSION AND CONCLUSIONS

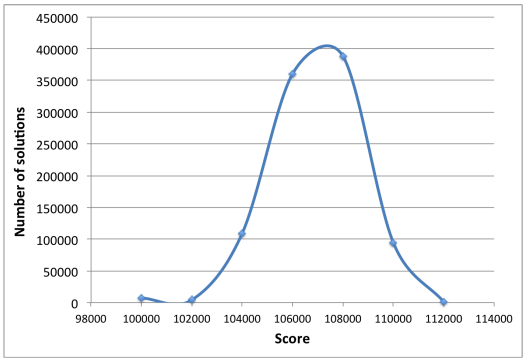
We have presented a design flow for cross architectural study that uses crowd sourcing to map benchmarks onto different architectures. We find that the crowd can reliably create good mappings, outperforming Simulated Annealing in 67 of 70 cases tested, and outperforming Simulated Annealing in all cases for the recommended architectures. Overall, we find 4 mesh architectures that are most promising in terms of energy and area: 4Way1Hop, 4Way1Hop I/O, 8Way, and 8Way I/O. We find StripeDR LC to be the dominant architecture for our benchmark suite among the stripe architecture variants. The recommended architectures are perceived by the crowd as presenting moderate mapping problems in terms of difficulty.



(a) Data Flow Graph.



(b) 4Way2Hops architecture.



(c) Number of solutions vs. Score.

Figure 22. There are hundreds of thousands of possible solutions within 5% of optimal for the problem dataf flow graph implemented on a 4Way2Hops architecture.

We can compare our findings with those in the literature. Several others ((Bansal et al., 2004; Mei et al., 2005; Bouwens et al., 2008)) echo our conclusions with regards to 4Way2Hop vs. 4Way1Hop. All conclude that the ability of 4Way2Hop to host more compact mappings is not sufficient to balance its high power requirements due to the greater interconnect.

We also studied the 4Way mesh architecture when exploring the less constrained architectures. In this architecture, a node has direct connections only to its direct horizontal and vertical neighbors. Consistent with previous research ((Bansal et al., 2004; Mei et al., 2005; Bouwens et al., 2008)) we found that the 4Way architecture consumed much more energy and area than the other variations. As observed by these other researchers as well, 4Way1Hop was always a significant improvement over the 4Way results. It is interesting to note that the 4Way architecture was also perceived by players to be the most difficult to map to. Players spent considerable time on this architecture, yet performed poorly relative to Simulated Annealing. Some players recognized that the difficulty was that they had to identify a planar embedding of the DFG in order to map to the 4Way architecture

without violations. One player states: “[In 4Way] the graph must be planar to get 0 violations (and if it is planar, 0 violations should always be possible)." Some players approached the problem by first identifying a planar embedding without regard to the overall width and height of the mapping, and then proceeded to map the graph compactly without introducing crossings. However, even when players could create successful mappings, the mappings were not favorable in terms of important aspects of performance, and we did not pursue this architecture farther when adding the more constrained I/O architectural variants.

Fewer authors appear to consider the 8Way architecture. da Silva and colleagues (Silva et al., 2006), however, conclude that 8Way has the worst performance by far of the architectures they have tested, in strong contradiction to our results. Without further investigation it is difficult to determine whether the source of the discrepancy is the mapping algorithms, the benchmarks, or other considerations. It is possible that a better mapping algorithm would have given different results in (Silva et al., 2006). We found, for example, that Simulated Annealing performed poorly on the 8Way architecture as compared to our players.

In conclusion, crowdsourcing appears a promising avenue to consider in the architectural design space exploration flow. It provides a designer very useful feedback from the game players that would be difficult to obtain from an automatic mapper. This extra information provides insights (e.g. what aspects of the architectures are creating difficult constraints and which architectures are relatively easy to map to) that can be valuable in selecting the best architectural solution for a particular application domain. We found it enlightening in uncovering deficiencies in existing mappers and opportunities for better architecture designs if only the mappers were improved.

Our future research goals include constructing a complete flow where architectural designs are sourced in real-time to players and results come back to the designer in real-time as they are generated. We would like to keep a line of communication open between designers and players. We would also like to expand our design case studies by allowing more variations in architectures (e.g. architectures with resource constraints, memories) that make the architectures more realistic and efficient. We plan to facilitate upload of user specified benchmarks. Our future plans also include studing scalability, exploring an effective approach for presenting large and difficult mapping problems to the crowd. We also plan to study how multiple players work in teams to solve larger and more difficult mapping problems. We have closed the loop from observed player strategies to automatic algorithms that also outperform Simulated Annealing (Mehta et al., 2013, 2015). We look forward to more research along these same lines as well.

ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation, under grants CCF-1117800 and CCF-1218656. We would like to thank Xiaozhong Luo, Emily Lofaro, Autumn Hood, Andrew Marin, Kiran Sanagapaty, and UNTANGLED players for their contributions to this project.

REFERENCES

Adya, S, Chaturvedi, S, Roy, J, Papa, D, and Markov, I. (2004). Unification of partitioning, placement and floorplanning. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*. 550 – 557.

Agnihotri, A and Madden, P. (2007). Fast Analytic Placement using Minimum Cost Flow. In *Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific*. 128 –134. DOI:http://dx.doi.org/10.1109/ASPDAC.2007.357974

Agnihotri, A. R, Ono, S, and Madden, P. H. (2005). Recursive bisection placement: feng shui 5.0 implementation details. In *Proceedings of the 2005 international symposium on Physical design (ISPD '05)*. 230–232.

Ambinder, M. (2009). Valve's Approach to playtesting: The application of empricisim. Game Developer's Conference. (2009).

Bansal, N, Gupta, S, Dutt, N, Nicolau, A, and Gupta, R. (2004). Network Topology Exploration of Mesh-Based Coarse-Grain Reconfigurable Architectures. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*. 10474.

Bauer, L, Shafique, M, and Henkel, J. (2009). Cross-architectural design space exploration tool for reconfigurable processors. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.* 958 –963.

Bertacco, V. (2012). Humans for EDA and EDA for humans. In *Proceedings of the 49th Annual Design Automation Conference*. ACM, 729–733.

Betz, V. (1997). VPR: A new packing, placement and routing tool for FPGA research. *Field-Programmable Logic and Applications* (1997), 1–10. http://www.springerlink.com/index/1673hwlmm7720606.pdf

Betz, V and Rose, J. (1997). VPR: A new packing, placement and routing tool for FPGA research. In *Field-Programmable Logic and Applications*. 213–222.

Bian, H, Ling, A. C, Choong, A, and Zhu, J. (2010). Towards scalable placement for FPGAs. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 147–156.

Bouwens, F, Berekovic, M, De Sutter, B, and Gaydadjiev, G. (2008). Architecture enhancements for the ADRES coarse-grained reconfigurable array. In *Proceedings of the 3rd international conference on High performance embedded architectures and compilers (HiPEAC'08)*. 66–81.

Bouwens, F, Berekovic, M, Kanstein, A, and Gaydadjiev, G (Eds.). (2007). *Architecture exploration of the ADRES coarse-grained reconfigurable array*. Springer.

Chen, T.-C, Hsu, T.-C, Jiang, Z.-W, and Chang, Y.-W. (2005). NTUplace: a ratio partitioning based placement algorithm for large-scale mixed-size designs. In *Proceedings of the 2005 international symposium on Physical design*. 236–238.

Chen, T.-C, Jiang, Z.-W, Hsu, T.-C, Chen, H.-C, and Chang, Y.-W. (2008). NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 27, 7 (July 2008), 1228 –1240. DOI:http://dx.doi.org/10.1109/TCAD.2008.923063

Choi, K. (2011). Coarse-Grained Reconfigurable Array: Architecture and Application Mapping. *IPSJ Transactions on System LSI Design Methodology* 4 (2011), 31–46.

Cong, J. (2011). Era of Customization and Implications to EDA. Keynote Speech, Synopsys University Reception, 48th Annual Design Automation Conference (DAC). (2011).

Cooper, S, Khatib, F, Treuille, A, Barbero, J, Lee, J, Beenen, M, Leaver-Fay, A, Baker, D, and Popović, Z. (2010)a. Predicting protein structures with a multiplayer online game. *Nature* 466 (August 2010). DOI:http://dx.doi.org/10.1038/nature09304

Cooper, S, Treuille, A, Barbero, J, Leaver-Fay, A, Tuite, K, Khatib, F, Snyder, A. C, Beenen, M, Salesin, D, Baker, D, and Popović, Z. (2010)b. The challenge of designing scientific discovery games. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games (FDG '10)*. ACM, New York, NY, USA, 40–47. DOI:http://dx.doi.org/10.1145/1822348.1822354

DeOrio, A and Bertacco, V. (2009). Human computing for EDA. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*. IEEE, 621–622.

Dimitroulakos, G, Georgiopoulos, S, Galanis, M. D, and Goutis, C. E. (2009). Resource aware mapping on coarse grained reconfigurable arrays. *Microprocessors and Microsystems* 33, 2 (March 2009), 91–105. DOI:http://dx.doi.org/10.1016/j.micpro.2008.07.002

Ebeling, C, Cronquist, D, and Franklin, P. (1996). RaPiD - Reconfigurable Pipelined Datapath. *Field-Programmable Logic Smart Applications, New Paradigms and Compilers* (1996), 126–135. http://www.springerlink.com/index/f45122367533h852.pdf

Ferreira, R, Garcia, A, and Teixeira, T. (2007). A polynomial placement algorithm for data driven coarse-grained reconfigurable architectures. *ISVLSI '07* (2007), 61–66. DOI:http://dx.doi.org/10.1109/ISVLSI.2007.14

Friedman, S, Carroll, A, Van Essen, B, Ylvisaker, B, Ebeling, C, and Hauck, S. (2009). SPR: an architecture-adaptive CGRA mapping tool. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. 191–200.

Fullerton, T. (2008). Game Design Workshop: A Playcentric Approach to Creating Innovative Games. Morgan Kaufmann, 1 edition. (2008).

Garey, M and Johnson, D. (1983). Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods* 4, 3 (1983), 312. DOI: http://dx.doi.org/10.1137/0604033

Gehring, S. W and Ludwig, S. H.-M. (1998). Fast integrated tools for circuit design with FPGAs. *FPGA '98* (1998), 133–139. DOI: http://dx.doi.org/10.1145/275107.275133

Geiger, D, Seedorf, S, Schulze, T, Nickerson, R, and Schader, M. (2011). Managing the crowd: towards a taxonomy of crowdsourcing processes. In *Proceedings of the 17th Americas Conference on Information Systems, Detroit*. 1–11.

Goldstein, S. C, Schmit, H, Budiu, M, Cadambi, S, Moe, M, and Taylor, R. (2000). PipeRench: A Reconfigurable Architecture and Compiler. *in IEEE Computer* 33, 4 (April 2000).

Hartenstein, R. (2001)a. A decade of reconfigurable computing: a visionary retrospective. *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001* (2001), 642–649. DOI:http://dx.doi.org/10.1109/DATE.2001.915091

Hartenstein, R. (2001)b. Coarse grain reconfigurable architecture (embedded tutorial). In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference (ASP-DAC '01)*. ACM, New York, NY, USA, 564–570.

Hartenstein, R, Herz, M, Hoffmann, T, and Nageldinger, U. (2000)a. KressArray Xplorer: a new CAD environment to optimize reconfigurable datapath array architectures. In *DAC*, Vol. 2. Ieee, 163–168. DOI:http://dx.doi.org/10.1109/ASPDAC.2000.835089

Hartenstein, R, Hoffmann, T, and Nageldinger, U. (2000)b. *Design-Space Exploration of Low Power Coarse Grained Reconfigurable Datapath Array Architectures*. Lecture Notes in Computer Science, Vol. 1918. Springer.

Hauser, J. R and Wawrzynek, J. (1997). Garp: A MIPS Processor with a Reconfigurable Coprocessor. In *IEEE Symposium on FPGAs for Custom Computing Machines*, Kenneth L. Pocek and Jeffrey Arnold (Eds.). IEEE Computer Society Press, Los Alamitos, CA, 12–21.

Karuri, K, Chattopadhyay, A, Chen, X, Kammler, D, Hao, L, Leupers, R, Meyr, H, and Ascheid, G. (2008). A Design Flow for Architecture Exploration and Implementation of Partially Reconfigurable Processors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 16, 10 (oct. 2008), 1281 –1294. DOI:http://dx.doi.org/10.1109/TVLSI.2008.2002685

Kennerly, D. (2003). Better Game Design through Data Mining. (15 Aug. 2003).

Kim, Y, Kiemb, M, Park, C, Jung, J, and Choi, K. (2005). Resource Sharing and Pipelining in Coarse-Grained Reconfigurable Architecture for Domain-Specific Optimization. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1 (DATE '05)*. 12–17.

Kim, Y, Mahapatra, R, and Choi, K. (2010). Design Space Exploration for Efficient Resource Utilization in Coarse-Grained Reconfigurable Architecture. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 18, 10 (oct. 2010), 1471 –1482. DOI: http://dx.doi.org/10.1109/TVLSI.2009.2025280

Kim, Y and Mahapatra, R. N. (2008). A New Array Fabric for Coarse-Grained Reconfigurable Architecture. In *DSD*. 584–591.

Krzemien, J, DeOrio, A, and Bertacco, V. (2011). FunSAT - Multi-Player. http://funsat.eecs.umich.edu/. (2011).

Lambrechts, A, Raghavan, P, Jayapala, M, Catthoor, F, and Verkest, D. (2008). Energy-Aware Interconnect Optimization for a Coarse Grained Reconfigurable Processor. In *VLSI Design, 2008. VLSID 2008. 21st International Conference on*. 201 –207.

Lanuzza, M, Perri, S, Corsonello, P, and Margala, M. (2007). A New Reconfigurable Coarse-Grain Architecture for Multimedia Applications. In *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*. 119 –126.

Lee, G, Lee, S, Choi, K, and Dutt, N. (2010). Routing-aware application mapping considering steiner points for coarse-grained reconfigurable architecture. *Reconfigurable Computing: Architectures, Tools and Applications* (2010), 231–243. http://www.springerlink.com/index/tj4437u3k735303v.pdf

Liang, C and Huang, X.-M. (2009). Mapping Parallel FFT Algorithm onto SmartCell Coarse-Grained Reconfigurable Architecture. In *ASAP*. 231–234.

Luo, T and Pan, D. Z. (2008). DPlace2.0: a stable and efficient analytical placement based on diffusion. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference (ASP-DAC '08)*. 346–351.

Malone, T. W. (1980). What makes things fun to learn? heuristics for designing instructional computer games. In *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*. ACM, 162–169. DOI:http://dx.doi.org/10.1145/800088.802839

Mao, A, Kamar, E, Chen, Y, Horvitz, E, Schwamb, M. E, Lintott, C. J, and Smith, A. M. (2013). Volunteering versus work for pay: Incentives and tradeoffs in crowdsourcing. In *First AAAI Conference on Human Computation and Crowdsourcing*.

Mehta, G, Crawford, C, Luo, X, Parde, N, Patel, K, Rodgers, B, Sistla, A. K, Yadav, A, and Reisner, M. (2013). UNTANGLED: A Game Environment for Discovery of Creative Mapping Strategies. *ACM Trans. Reconfigurable Technol. Syst.* 6, 3, Article 13 (October 2013), 26 pages. DOI:http://dx.doi.org/10.1145/2517325

Mehta, G, Jones, A. K, Stander, J, Baz, M, and Hunsaker, B. (2009). Interconnect Customization for a Hardware Fabric. *ACM Transactions on Design Automation for Electronic Systems (TODAES)* 14, 1 (January 2009).

Mehta, G, Patel, K, Parde, N, and Pollard, N. S. (2013). Data-Driven Mapping Using Local Patterns. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 32, 11 (2013), 1668–1681. DOI:http://dx.doi.org/10.1109/TCAD.2013.

2272541

Mehta, G, Patel, K, and Pollard, N. S. (2015). On fast iterative mapping algorithms for stripe based coarse-grained reconfigurable architectures. *Special Issue on Reconfigurable and Adaptive Computing , International Journal of Electronics* 102, 1 (2015), 3–17.

Mei, B, Lambrechts, A, Mignolet, J.-Y, Verkest, D, and Lauwereins, R. (2005). Architecture exploration for a reconfigurable architecture template. *Design Test of Computers, IEEE* 22, 2 (march-april 2005), 90 – 101. DOI:http://dx.doi.org/10.1109/MDT.2005.27

Michelucci, P. (2013). *Handbook of human computation*. Springer.

Mirsky, E and Dehon, A. (1996). MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources. In *Proc. of IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*.

Miyamori, T and Olukotun, K. (1998). REMARC: Reconfigurable Multimedia Array Coprocessor. In *IEICE Transactions on Information and Systems E82-D*. 389–397.

NSF, . (2013). Winners of the 2012 International Science & Engineering Visualization Challenge. (2013).

Quinn, A. J and Bederson, B. B. (2011). Human computation: a survey and taxonomy of a growing field. In *Proceedings of the 2011 Annual Conference on Human factors in computing systems*. ACM, 1403–1412.

Ramey, J, Boren, T, Cuddihy, E, Dumas, J, Guan, Z, van den Haak, M. J, and De Jong, M. D. T. (2006). Does think aloud work?: how do we know?. In *CHI '06 extended abstracts on Human factors in computing systems (CHI EA '06)*. ACM, New York, NY, USA, 45–48. DOI:http://dx.doi.org/10.1145/1125451.1125464

Sankar, Y and Rose, J. (1999). Trading quality for compile time: ultra-fast placement for FPGAs. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays (FPGA '99)*. ACM, 157–166. DOI:http://dx.doi.org/10.1145/296399.296449

Science, . (2013). 2012 International Science & Engineering Visualization Challenge. http://www.sciencemag.org/site/special/vis2012/. (2013).

Sechen, C and Sangiovanni-Vincentelli, A. (1985). The TimberWolf placement and routing package. *Solid-State Circuits, IEEE Journal of* 20, 2 (april 1985), 510 –522. DOI:http://dx.doi.org/10.1109/JSSC.1985.1052337

Sharma, A, Hauck, S, and Ebeling, C. (2005). Architecture-adaptive routability-driven placement for FPGAs. In *Field Programmable Logic and Applications, 2005. International Conference on*. 427–432. DOI:http://dx.doi.org/10.1109/FPL.2005.1515759

Silva, M. V. D, Ferreira, R, Garcia, A, and Cardoso, J. M. P. (2006). Mesh Mapping Exploration for Coarse-Grained Reconfigurable Array Architectures. In *Reconfigurable Computing and FPGA's, 2006. ReConFig 2006. IEEE International Conference on*. 1 –10. DOI:http://dx.doi.org/10.1109/RECONF.2006.307749

Singh, H, Lee, M.-H, Lu, G, Kurdahi, F. J, Bagherzadeh, N, and Filho, E. M. C. (2000). MorphoSys: An Integrated Reconfigurable system for data-parallel and computation-intensive applications. *IEEE Trans. Comput.* 49, 5 (2000), 465–481.

Sistla, A, Parde, N, Patel, K, and Mehta, G. (2013). Cross-Architectural Study of Custom Reconfigurable Devices Using Crowdsourcing. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*. 222–230. DOI:http://dx.doi.org/10.1109/IPDPSW.2013.133

Sunstein, C. R. (2006). *Infotopia: How many minds produce knowledge*. Oxford University Press, USA.

Surowiecki, J. (2005). *The wisdom of crowds*. Random House Digital, Inc.

Taghavi, T, Yang, X, and choi, B.-K. (2005). Dragon2005: Large-scale mixed-size placement tool. In *Proceedings of the 2005 International Symposium on Physical design (ISPD '05)*. 245–247.

Taylor, M, Psota, J, Saraf, A, Shnidman, N, Strumpen, V, Frank, M, Amarasinghe, S, Agarwal, A, Lee, W, Miller, J, Wentzlaff, D, Bratt, I, Greenwald, B, Hoffmann, H, Johnson, P, and Kim, J. (2004). Evaluation of the Raw microprocessor: an exposed-wire-delay architecture for ILP and streams. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*. 2 – 13.

Tehre, V and Kshirsagar, R. (2012). Survey on Coarse Grained Reconfigurable Architectures. *International Journal of Computer Applications* 48, 16 (June 2012).

Terry, L, Roitch, V, Tufail, S, Singh, K, Taraq, O, Luk, W, and Jamieson, P. (2009). Harnessing Human Computation Cycles for the FPGA Placement Problem.. In *ERSA*, Vol. 9. 188–194.

The Assembly Line, . (1989). Pipe Mania. (1989).

Theodoridis, G, Soudris, D, and Vassiliadis, S. (2008). A Survey of Coarse-Grain Reconfigurable Architectures and Cad Tools. In *Fine- and Coarse-Grain Reconfigurable Computing*, Stamatis Vassiliadis and Dimitrios Soudris (Eds.). Springer Netherlands, 89–149.

Viswanathan, N and Chu, C.-N. (2005). FastPlace: efficient analytical placement using cell shifting, iterative local refinement,and a

hybrid net model. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 24, 5 (2005), 722–733.

Von Ahn, L, Blum, M, and Langford, J. (2004). Telling humans and computers apart automatically. *Commun. ACM* (2004), 56–60. DOI:http://dx.doi.org/10.1145/966389.966390

Von Ahn, L and Dabbish, L. (2004). Labeling images with a computer game. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '04)*. ACM, 319–326. DOI:http://dx.doi.org/10.1145/985692.985733

Von Ahn, L and Dabbish, L. (2008). Designing Games with a Purpose. *Commun. ACM* 51, 8 (August 2008).

Von Ahn, L, Ginosar, S, Kedia, M, Liu, R, and Blum, M. (2006)a. Improving accessibility of the web with a computer game. In *Proceedings of the SIGCHI conference on Human Factors in computing systems (CHI '06)*. ACM, 79–82.

Von Ahn, L, Liu, R, and Blum, M. (2006)b. Peekaboom: A Game for Locating Objects in Images. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 55–64. DOI:http://dx.doi.org/10.1145/1124772.1124782

Webster, J. (1988). Making computer tasks at work more playful: Implications for systems analysts and designers. In *Proceedings of the ACM SIGCPR conference on Management of information systems personnel (SIGCPR '88)*. ACM, 78–87. DOI:http://dx.doi.org/10.1145/57216.57227

Yoon, J, Shrivastava, A, Park, S, Ahn, M, and Paek, Y. (2009). A Graph Drawing Based Spatial Mapping Algorithm for Coarse-Grained Reconfigurable Architectures. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 17, 11 (2009).