NTNU
Norwegian University of
Science and Technology

TDT4900 Computer Science, Master's Thesis

Optimization of Seed Selection for Information Diffusion with High Level Synthesis

Julian Lam

Spring 2016

Department of Computer and Information Science
Norwegian University of Science and Technology

Supervisor 1: Donn Alexander Morrison
Supervisor 2: Yaman Umuroglu

Abstract

Information Diffusion is often used for different simulations in network research because it simulates how information propagates thorough a network. Measuring spread and speed, we can find influential targets in the network. Such targets are optimal targets to pass message during disaster scenario, vaccinate to prevent spreading of a disease, or even targets for viral marketing.

High Level Synthesis (HLS) have in recent years matured greatly. With HLS, designing custom architectures is no longer a dream.

Assignment

Information diffusion is a field of network research where a message, starting at a set of seed nodes, is propagated through the edges in a graph according to a simple model. Simulations are used to measure the coverage and speed of the diffusion and are useful in modelling a variety of phenomena such as the spread of disease, memes on the Internet, viral marketing and emergency messages in disaster scenarios.

The effectiveness of a given spreading model is dependent on the initially infected nodes, or seeds. Seed selection for an optimal spread is an NP hard problem and is normally approximated by selecting high-degree nodes or using heuristic methods such as discount-degree or choosing nodes at different levels of the k-core.

High-level synthesis (HLS) is becoming an important tool in the optimization/acceleration of algorithms in hardware. Starting with an algorithm written in a high-level language such as C or C++, HLS aids with hardware design by providing a methodology and tools that guide the developer through the design process.

This project should employ HLS as a design methodology for hardware accelerated seed selection in large graphs. The student will study seed selection for a given diffusion model, write a high-level model, and use HLS to implement a hardware design that exploits parallelism in the seed selection algorithm in order to improve performance over a GPCPU implementation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Information Diffusion is a field in network research where a message, or data, is propagated through a *network* or a *graph*. The message originates from a chosen set of nodes, known as *seed nodes*. These seed nodes passes the message to its neighbours through the edges and thus propagates the message over the entire network. The effectiveness of the Diffusion is measured through the spread and the speed of propagation and is dependent on the chosen seed nodes. By finding the most optimal set of seed node, we can potentially stop an epidemic by vaccinating influential nodes, we can find important target for viral marketing by giving free sample, and use this information to quickly spread message during disaster scenarios[2] [3].

There are multiple studies done regarding information diffusion, [4], [2], [5], [6]. But as far as we know, there are none that focus on optimizing the seed selection in hardware. The current seed selection algorithm is an greedy solution[7], where every set of nodes is tested and the set with best coverage and time is chosen. This is a time consuming process and highly parallelizable, which makes it a good candidate for *Field-programmable gate arrays*(FPGAs).

High Level Syntesis (HLS) transform high level behaviour and constrains to lower level design.[8]. It makes it possible to implement an algorithm in high level language, C or C++, and generate an optimal design in *verilog* or *VHDL*. Verilog and VHDL are hardware descriptive languages designed to describe digital systems [9].

Unlike traditional hardware design, HLS allows programmers with limited knowledge of hardware design to create an optimal custom *Intellectual property core*(IP-core). In HLS, programmers can test out different optimization schemes

in a short period of time, thus reducing development overhead.

In this thesis, we have implemented a simple IP-core that performs information diffusion using the *Independen Cascade model* (ICM) as *Breadth-First Search* (BFS) over boolean semirings. This is done by using HLS as the development tool.

1.2 Assignment Interpretation

From the assignment text, these task were chosen as the main focus of this thesis:

Task 1 ***(mandatory)*** Implement Information Diffusion as Sparse matrix vector multiplication, with high level language C.

Task 2 ***(mandatory)*** Tailor the implementation of Information Diffusion for synthesise with Vivado HLS.

Task 3 ***(optional)*** Implement said design on a Zynq FPGA board.

Task 4 ***(optional)*** Extend the system to be able to handle graph in the size of toy graphs(containing 2^{26}) nodes)

1.3 Report Structure

We have here the basic outline for this report and a short overview of the remainder of this report:

Chapter 2: Background contains the theory regarding network, information diffusion, matrix vector multiplication and high level synthesis.

Chapter 3: Related Work gives a short introduction of the state of the art of HLS implementations, information diffusion research and different optimization of BFS.

Chapter 4: Design and Implementation present our implementation of our IP-core and give a brief introduction regarding HLS implementation and optimization.

Chapter 5: Result and Discussion will compare the result our core gener-

ated compared to a C-simulation. We will also discuss some of the design choices regarding the IP-core.

Chapter 6: Future Work present how our design can be further improved.

Chapter 7: Conclusion provides a concluding remarks regarding this paper and a summary of the identified tasks.

Chapter 2

Background

In this chapter, we will look at the fundamental concepts and theory of the different diffusion models, seed selection algorithms and perform BFS over the boolean semiring. We will also have a look at HLS and specifications of the Zedboard. This chapter will contain notations that we will use throughout the report.

We will look at the independent cascade model, which is a special case of breadth first search [10]. By looking at how to improve BFS, we can apply such optimization to ICM and the seed selection algorithm.

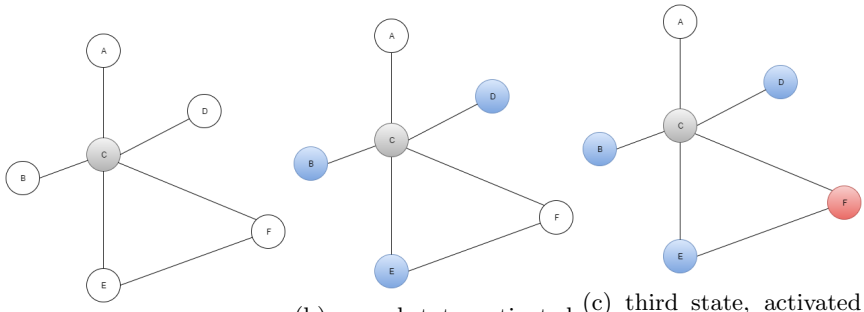
2.1 Information Diffusion

Information diffusion is looking at how information is propagated through a network. A vertex can be either activated(infected) or inactivated(healthy/noninfected), each node can spread the contagion(activation,infection) to their neighbour. Some examples would be how a meme, a trend or a disease is spread through a community. The process consist of a set of starter nodes, which we will call seed nodes, that are "infected" at initial time step. During each time step, there are a percentage p_g where the "infected" nodes would "infect" its neighbours. Seed nodes is a set of k nodes that in the initial time-step are infected. They will pass on the information/infection during each time-step and the information/infection will propagate through the network.

2.2 Basic Diffusion Models

For information diffusion, two common models are used for simulations. Those are the *linear threshold model*(LTM) and the *independent cascade model*(ICM) [11].

ICM is a model where each spread of contagion is dependent on a "coin flip". Each person has a chance to be contaminated, and during each diffusion, dependent on the result of the coin flip, that person is either contaminated/infected or healthy. In ICM, an infected person can not reinfect the previously spared person. The probability of infection can be either globally, or locally. In Figure 2.1a, we can see a situation where C have five neighbour,A,B,D,E and F. In the initial state, only C is infected. Five separate "coin flips" was done and resulted in three more activations as shown in Figure 2.1b. C spread the contagion to three other people, while A and F were spared. In Figure 2.1c F is activated by E.



(a) Initial state

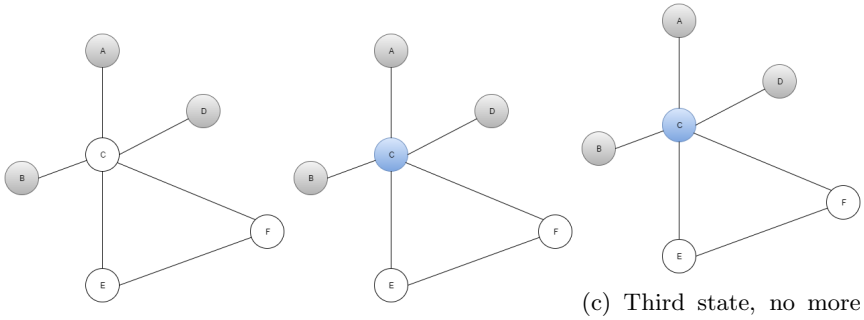
(b) second state, activated by C is marked blue

(c) third state, activated by E is marked red, no more activation from C

Figure 2.1: Independent cascade model

For the LTM activation is not dependent on a probability and a "coin flip", but a internal threshold of activated neighbours. The spread of contagion is dependent on how large of a fraction of their neighbour is activated. For our example, let say that for a person to be infected, 0.6 of their total neighbour must be infected before they can contract the contagion. In Figure **??**, we see that in the initial state A,B and D are infected. For C, more then 0.6 of its total neighbours are activated, this resulted in an activation of it too as seen in Figure 2.2c. We can see that both E and F have only 0.5 of their neighbours activated, and thus in Figure 2.2c no more activation is presence.

A real world example of the LTM would be e.g. A new product on the marked. People would adopt the new product it enough of their acquaintance is



(a) Initial state, A,B,D is activated.

(b) Second state, S is activated and marked blue.

(c) Third state, no more activation, since both E and F have less then 0.6 activated neighbours.

Figure 2.2: Linear Threshold mode

using it (activated). While ICM would be more directed marketing. Where free samples are given to some users. Those users would then promote the product and potentially spread the product to other.

BFS is a special case of ICM, both algorithm traverse graphs in a level by level approach. The difference is that bfs will activate the visited vertices, while ICM depends on the probability.

2.3 Breadth First Search

BFS is a tree traversal algorithm. BFS start at the root node v_r. The algorithm then stores all v_rs children nodes in a *queue*. The algorithm then takes the first node from the queue, v_1 and stores all the children nodes to v_1 in the back of the queue. This process continues until the queue is empty and all the nodes have been iterated over.

BFS is a common graph iteration algorithm, but is often limited by the irregular memory access where the algorithm have to find the data stored in different spaces in memory.

2.3.1 BFS to Data Diffusion

The motivation for transforming breadth first search as matrix-vector multiplication is that displaying the graph algorithm as a matrix multiplication can display the data access pattern for the algorithm and can be readily optimized [12].

As mentioned before, ICM is a special case of the breadth first search. By modifying the algorithm proposed earlier, we can in theory perform ICM with

Algorithm 1 Breadth First Search

```
1: dist[∀v ∈ V] = −1; currentQ, nextQ = ∅
2: step = 0; dist[root] = step
3: ENQUEUE(nextQ,root)
4: while nextQ ≠ ∅ do
5:     currentQ = newxtQ; nextQ = ∅
6:     step = step + 1
7:     while currentQ ≠ ∅ do
8:         u = DEQUEUE(currentQ)
9:         for v ∈ Adj[u] do
10:            if dist[v] == −1 then
11:                dist[v] = step
12:                ENQUEUE(nextQ, v)
   return dist
```

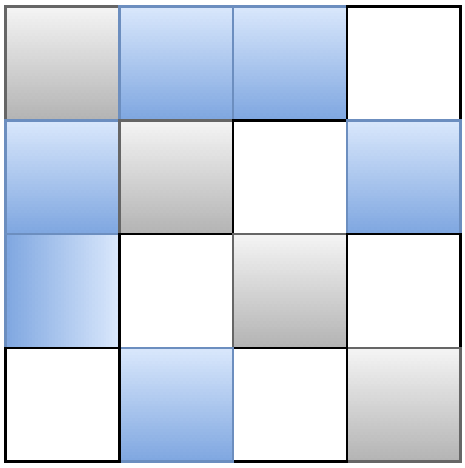
matrix-vector multiplication.

2.4 Matrix Notations

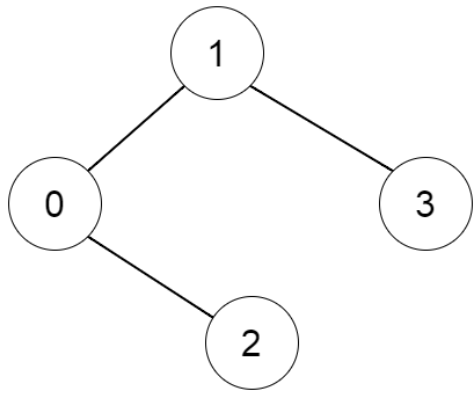
Nodes and edges are not the only way to represent a network. Networks can be represented as *sparse adjacency matrices* [12] [13]. By representing networks as sparse matrix, we can often discover different ways to optimize the algorithm, or a different structure to store the data. The adjacency matrix in particular, is a interesting way to represent the graph. A graph $G = (V, E)$, G have N vertices and M edges, this correspond to a N×N adjacency matrix called A. If A(i,j)=1, then there is an edge from v_i to v_j. Otherwise its 0. In Figure:**??**, we can see how a undirected graph can be represented as an adjacency matrix. Each square symbolise a connection between two nodes. To generate a undirected graph as a adjacency matrix, the matrix must be mirrored diagonally, meaning if A(i,j)=1, then A(j,i)=1, if this is not true, then the matrix would be representing a directed graph.

2.4.1 Sparse Matrix

A sparse matrix is a matrix containing few nonzero. Social graph with few edges would often be represented as a sparse matrix. Since sparse matricies only have few non-zero elements, by storing only the non-zero elements, we can have savings in memory.



(a) The adjacency matrix



(b) The graph corresponding to the adjacency matrix

Figure 2.3: Sparse matrix to graph

2.4.2 Breadth First Search as Matrix Multiplication.

From [12], we can see that BFS can be recast as algebraic operations. BFS can be performed by applying matrix-vector multiplication over Boolean semirings [10]. The graph is represented as a adjacency matrix A, then for the root node, a vector x(root)=1 is multiplied with the matrix A. $A \times x_0 = y_0$. y_0 is the result after the first matrix-vector multiplication and in the next iteration, $x_1 = y_0$. We can see from the Figure2.4

2.4.3 Semiring

A *semiring* is a set of elements with two binary operations. The two operations are often known as "addition"(+) and "multiplication"(×). As we shown in previous section, the algorithm perform matrix multiplication uses the two operations, multiplications and addition. In [10], the AND and OR operator was chosen instead of the normal addition and multiplication.

2.5 Seed Selection Algorithm

The seed selection algorithm, is the algorithm used to select the initial k seed nodes to be chosen at the start of the information diffusion. Each selected nodes is in the initial timestep activated. During each timestep, the seed nodes will

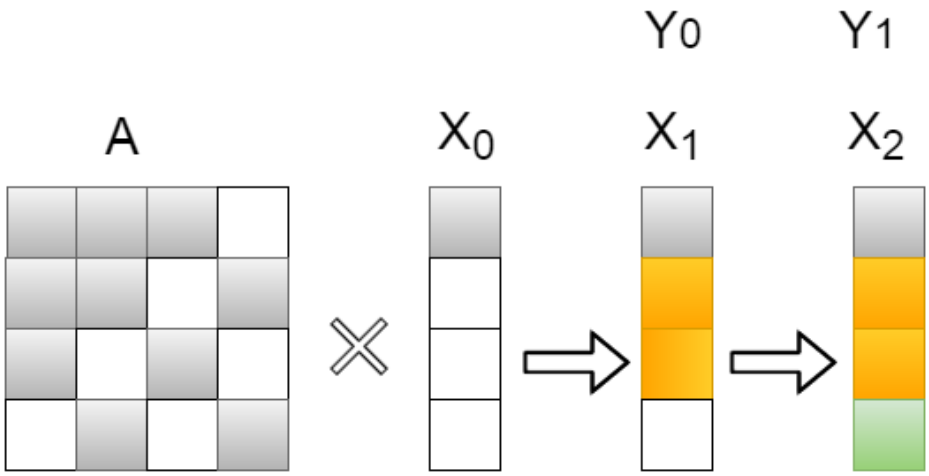


Figure 2.4: BFS on Boolean semiring

propagate the activation along the network depend on what diffusion model is used. We can compare it to a new gadget or a cosmetic company trying to promote a new product. By selecting a few influential persons to give a free sample, the new trend would most likely spread through *viral marketing* [14]. The seed selection algorithm would be the algorithm to select the few influential individuals to receive this free sample. There are multiple different scheme to choose from, in this section, we will focus on four different algorithms, greedy algorithm, degree algorithm, random algorithm and the independent greedy algorithm.

2.5.1 The Greedy Algorithm

The greedy algorithm [7] [15] proposed by Kempe et al, is known to be the best algorithm according to the result from [7]. The greedy algorithm starts by itterate over the entire network and finds one node that have the largest coverage and stores that node in the set S. The algorithm then activates that node from A, and computes the coverage of each node in the network with the previous chosen node. After the algorithm finds the second node, that node is stored in A with the previous node and we have now chosen two seed node. The algorithm continues until we have chosen k seed nodes, where each have been tested to have maximum coverage in relation with the previous chosen nodes. To compute the maximum coverage, the algorithm have to test every combination of nodes together, this results in heavy computation and the algorithm would therefore not scale well.

We can look at the greedy algorithm as a special case of BFS with a transistion

probability. Each node in the network will be chosen as the seed node for the

Algorithm 2 Greedy Algorithm

1: Start with $A = \emptyset$
2: **while** $|A| \leq l$ **do**
3: For each node x, use repeated sampling to approximate $\sigma(A \cup x)$ to within $(1 \pm \varepsilon)$ with probability 1δ
4: Add the node with largest estimate for $\sigma(A \cup x)$ to A.
5: Output the set A of nodes.

2.5.2 The Degree Algorithm

Another popular algorithm is the degree algorithm [7]. Unlike the greedy algorithm, does not compute the coverage of node, the algorithm picks the top k nodes according to the degree distribution instead. The node chooses the top k nodes with the highest degree and stores them as the seed nodes. This approach benefits over the greedy algorithm by not having as much computation time as the greedy algorithm since only one itteration is needed to compute the degree to node. The disadvantage is that this algorithm does not take the degree correlation into acount. As mentioned in section **??**, high degree nodes would often have common node as neighbor. This would result in multiple overlapping activated node choosen.

Algorithm 3 Degree Algorithm

1: Start with $A = \emptyset$
2: **while** $|A| \leq l$ **do**
3: For each node x, use repeated sampling to compute DegreeMax(x).
4: Add the node with largest degree to A.
5: Output the set A of nodes.

2.5.3 Independent Algorithm

Another algorithm is the independent greedy algorithm. The algorithm iterates through the network, computing the spread of each node. The algorithm then chooses the vertex with the largest coverage independent of the other previous chosen nodes. This algorithm is a special case of the greedy algorithm mentioned above.

Algorithm 4 Independent Algorithm

1: Start with $A = \emptyset$
2: **while** $|A| \leq l$ **do**
3: For each node x, use repeated sampling to approximate $\sigma(A \cup x)$ to within $(1 \pm \varepsilon)$ with probability 1δ
4: Add the node with largest estimate for $\sigma(x)$ to A.
5: Output the set A of nodes.

2.5.4 Random Algorithm

The last one is the random algorithm. The random algorithm just picks a random seed node. This approach is the simplest to implement and easiest. The downside is that this is random and there are no strategic choosing of seed node.

2.6 High Level Synthesis

High Level Synthesis converst algorithms implemented on higher level down to *Register Transfer Level* (RTL)[16]. RTL is models of digital circuits displaying flow of data between register, logical operations and such. It is commonly used to descripe low level digital systems. HLS is known to be able to reduce development effort and cost of creating specialized hardware compared to traditional hand-drawn RTL designs[17][18][16]. By taking high level language such as C, C++ and SystemC implementations and generate the optimal architecture. HLS allows the user to generate custom *Intellectual Property*(IP) core. An IP-core is a custom created data core that have an output and an input port.

2.7 ZedBoard

The Zedboard that we used for this project, is *Xilinx Zynq-7000 All programmable System-on-chip(SoC)* Z-7020. Consist of a dual core *ARM cortex-A9 MPCore* based processing system(PS) and an *Artix-7 XC7Z020* FPGA. The FPGA is the *programmable logic*(PL). The Zedboard have 512 MB DDR3 RAM, 256MB Quad-SPI Flash and 4GB SD card.[19]. The system offers the flexibility and scalability of an FPGA[20].

THe FPGA use *Advance eXtensible Interface*(AXI4)bus protocol. There are three types if AXI4 interfaces:

- **AXI4 Lite**- Simple, memory mapped communication. Useful for small single read.

- **AXI4-Stream** - for continues streaming of data.
- **AXI4** - For memory mapped applications.

A component with PL implemented would be able to connect to the PS through a AXI4 bus port. The close coupling between t

2.8 RMat

One problem during graph analyzation and calculation is finding suitable graphs to analyses. Generate graphs with desired properties is not easy to do. One solution proposed by Chakrabartiy et al is to use the "recursive matrix" or R-mat model. The R-mat model generates graph with only a few parameters, the generated graph will naturally have the small world properties and follows the laws of normal graphs, and have a quick generation speed [1]. The R-mat models goal is to generate graphs that matches the degree distribution, exhibits a " community " structure and have a small diameter and matches other criteria. [1]. The algorithm to generate such a recursive matrix is as follows: The idea is to partition the adjacency matrix into four equally sized part branded A,B,C,D, as shown in Figure2.5. The adjacency matrix starts by having all element set to 0. Each new edge is "dropped" onto the adjacency matrix. Which section the edge would be placed in, is chosen randomly. Each section have a probability of a, b, c, d, and $a + b + c + d = 1$. After a section is chosen, the partition that was chosen is partitioned again. This continues until the chosen section is a 1x1 square and the edge is dropped there. From the algorithm, we can see that the R-mat generator are capable to generate graphs with total numbers of node $V = 2^x$. Since the algorithm partitioned the matrix into four part. This is approach would only generate a directed graph. To generate undirected graph, $b = c$ and the adjacency matrix must make a "copy flip" on the diagonal elements, like Figure 2.5.

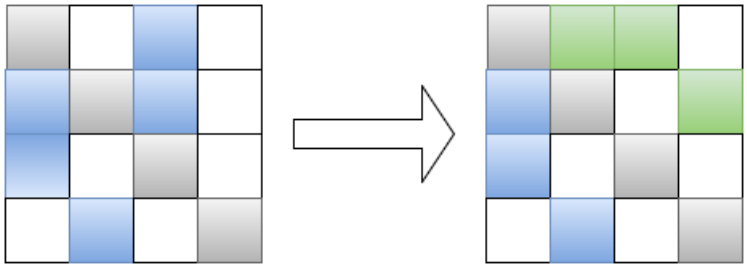


Figure 2.5: How the adjacency matrix is flipped on the diagonal

Chapter 3

Related Work

In this chapter, we will look at the state of research regarding High Level Synthesis, information diffusion, and optimization of independent cascade model and breadth-first search.

3.1 Information Diffusion

There are multiple studies done regarding information diffusion. One study shows how information diffusion can be applied during an disease outbreak[2], viral marketing[14], coordinat during crisis situation[21].

Different models of information diffusion have been done on blogs[22][23], and twitter[24]. We can see that in an age of social media, the studies of information diffusion is more relevant then ever.

While [6] have argued that the emerging of social networks and media, have changed the traditional model. The activation is no longer only relying on neighbour nodes, but also an external influence. They found that large amount of information volume in Twitter is the result of network diffusion, while a small amount is due to external events and factors outside the network[6]. Another studie shows that during the 2011 Egyptian Uprising, how a large amount of rebel movement were "tweeted"[21] during the uprising.

As we mentioned in Chapter 2, we mainly focus on two common information diffusion models, ICM and LTM. But there are different models too. [5] proposed several different problems with traditional models where each node is either *activated*(infected, influenced, '1') or *inactive*(healthy, not reached, '0'), and passes the *contagion*(information, data, infection, influence) to neighbouring nodes through the edges. The report mentioned different assumptions that such models make. Among them is that a complete graph is provided, the spread of

contagion is from a known source, and that the structure in the network is sufficient to explain the the behaviour[5]. The report propose an alternative model, *Linear Influence Model*(LIM), where the focus is on the global influence that an infected node has on the rate of diffusion through the implicit network. This model makes the assumption that newly activated nodes are dependent on previous activated nodes. The LIM does not need explicit knowledge of the entire network, instead the model takes the newly activated nodes and model them as a *influence function*, which is used to find the global influence.

3.2 High Level Synthesis

High Level Synthesis as a concept have been around since the mid-1980s and early-1990s[16][25]. Carnegie-Mellon University design automation (CMU-DA)[26][27] was a pioneering early version of HLS tools. The tool gathers quickly considerable interest. A number of HLS tools were built in later years mostly for prototyping and research[28][29][30]. Some of these was able to produce real chips, but the reason for lack of further development and adaptation, was that RTL synthesis was not a widely accepted and a immature field. This often lead to suboptimal solutions.

Around the year 2000, new HLS tools was developed in academia and in the industry. These tools, used hihg level language, C and C++. Vivado HLS, designed by Xilinx [31], is one such HLS tool. The Vivado HLS became free during their 2015.4 update[32]. This resulted in an revived interest in HLS. The community around HLS is also evolving, on the Xilinx-forum, there are multiple answers and active members. We can see that the solution designed by HLS tools is close to traditional hand-crafted designs[33].

[16] goes into the history of HLS throughout the ages.The paper discuss different problems and reason for failed commercialization of early HLS tools, Cong et al, concluded that the problem with the early HLS tools can be summarized by the following reasons: *Lack of comprehensive design language support*, *Lack of reusable and portable design specification*, *Narrow focus on datapath synthesis*, *lack satisfactory QoR*, *Lack of a compelling Reason/event to adopt new design methodology*. Early version of HLS tools was not a C-to-RTL transformation. Most of them needed a custom *Hardware Descriptive Language*(HDL). Lacking of reusable and portable design specification resulted in that HLS tools required user to include detailed information regarding timing and interface information into source code. This resulted in a target dependent solution and can't easily port to other devices. The narrow focus on datapath synthesis resulted in a lack of focus on interface to other hardware modules and platform integration. Those aspect was left to the users to solve system integration problem. The lack of foundation to accurately measure HLS result and often failed ot meet timing and

power requirement with early HLS tools were another limiting factor. The last reason, was that there where no real driving force to turn developers over to such a young and early development format. HLS tools shows interesting capabilities, but most developers did not want to move from the safe and tested RTL design methodology. The paper concludes with that current(2011)HLS tools showed huge potentials in becoming standard in selected deployment.

3.2.1 Applications using HLS

In [34], HLS was used to design an accelerator for database analytic and SQL operation. The design was implemented on a Virtex-7 xc7vx690t-g1761-2 FPGA with focus on accelerating operations as join, data filter, sort, merge and string matching. The accelerator was implemented in C++ in Vivado HLS and optimized with UNROLL directive, PIPELINE directive and ARRAY_PARTITION. The UNROLL directive unroll all of the specified loops, while the PIPELINE directive allows multiple accelerator to process data at each clock cycle. The ARRAY_PARTITION directive partition data into registers. The accelerator showed promises, giving a 15-140× speedup compared to Postgres software DBMS running selected TPCH queries.

[35] explored the advanteage and disadvanteges of HLS implementation of image processing. They argued that custom algorithms on FPGA platforms will most likely result in an improvement, but the algorithms must be tailored to the platforms. The author conducted different case studies to show both the strengths and the weaknesses of HLS. The report goes through image filtering, connected component analysis and two dimensional fast-Fourier transformation(FFT). One example that the author brings up, is during image filtering where HLS was not able to identify the standard accessing pattern during special cases. This resulted in that the HLS built additional hardware to counter such a exception. The report concluded that while HLS can significantly reduce development time and improve utilization of the design space, it is still important to focus on careful design. The report concludes that HLS can offer many benefits, and is an improvement over conventional RTL-designs, but is not an replacement for hardware designers or clever designs.

[36] implemented an fast Fourier transform (FFT) algorithm for different digital system processing application in HLS. There the authors used Simulink for verification of design, and implemented it in HLS.

[18] discuss improvements to the current HLS tools with polyhedral transformation. Here they discuss a problem with HLS, which is that unless the code is inherently compatible, HLS can not apply most of the optimizations. Zuo et al. proposed the polyhedral model, the model takes data dependent multi-block program as input and performs three steps: Classification of array access pat-

terns, performance Metric, and implementation. During the classification of array access patterns, a set of data access pattern is defined and classified. Then the appropriate loop transformation is applied. The next step, the performance of each loop transformation with data-dependency is estimated, and the best improvement is chosen. In the final step, the chosen solution, loop transformation and inserting HLS directive is applied. Then a interface block for the data-dependent blocks is generated. The generated communication block is then optimized depending on how it behaves. The paper concludes with that the polyhedral model can model can find important loop transformation, thus enable optimization such as pipeline and parallelization.

[33] is a case-study where HLS is used to implement two compute heavy machine learning techniques with different computational properties. The two algorithms that was tested was *Lloyd's Algorithm* and a *Filtering Algorithm*. The result was that for the first case, a similar performance between the HLS solution and a hand-written solution, while the second algorithm was severely worse with HLS, if the developer did not customized for HLS.

3.3 Different optimization scheme

There are a large amount of different optimization research on graph traversal, especially on Breadth First Search.

[10] proposed a hybrid FPGA-CPU heterogeneous platform for BFS. The idea is to run the first couple of steps on the CPU core, then switch over to the FPGA accelerator to explore the rest of the graph. The CPU is better suited for calculating while there is a smaller frontier, while the FPGA core is better suited for a larger frontier. By exploiting the characteristics of small-world networks where the frontier is much larger after two-three iterations, one can significantly improve computation time. The report proposed an alternative method of performing breadth-first search. By performing it as a sparse matrix vector multiplication over a boolean semiring, more parallilization option was discovered. The result was a speedup of 7.8 compared to a pure software implementation ,and 2$\times$ better compared to an accelerator only implementation.

[37] propose a hybrid solution, combining a conventional top-down algorithm and a novel bottom-up algorithm. The optimization in this paper focuses on examining fewer edges, thus reduce computation time and trying to circumvent one major drawback with BFS; memory-bound on shared memory. The top down approach is the traditional algorithm, where a frontier expands and visits all nodes on that level, before each node checks its neighbour for unvisited vertices. Unvisited vertices is placed in the frontier and marked as visited. The bottom up algorithm, contrary to the top down algorithm, is where each children vertex tries to find a potential parent. A neighbour vertex can be parent if the neighbour is

also in the frontier. This results in that after a vertex finds its parent, there is no need to traverse the rest of the frontier. By using different approach during different time, the report was able to achieve a speedup of 3.3 - 7.8× on synthetic graphs and 2.4 4.6× on real social network graphs.

[38]

Chapter 4

Design and Implementation

In this chapter, we will present our implementation of the sparse matrix vector multiplication (SpMV) over boolean semiring, how we implemented with HLS and how we connected the IP-cores in the FPGA.

4.1 IP-core

For this project, we have implemented an IP-core with Vivado HLS. The IP-core applies a modified version of SpMV over a adjacency matrix and a set of vertices known as seed nodes. We will givee a brief introduction to HLS programming and how to customize the algorithm for HLS.

4.1.1 IP-core

Our implementation of the SpMV is done in HLS. The algorithm takes in an adjacency matrix and a set of seed nodes as input, and outputs a result vector showing which nodes were activated. The algorithm stops when all the nodes have been activated or it is unable to find any new candidates. Unlike normal SpMV, where each iteration will activate all their neighbours, an ICM is dependent on random number generator (RNG) and a global or local probability. For this project a global probability of 5% was used, i.e. each node had a 5% chance of being activated. If v_1 was not activated by V_r on the first iteration, V_r can not reactivate v_1 on the next iteration. To prevent a reactivation, as mentioned above, a frontier vector will be sent in instead of the result from previous iteration.

A vector is a list of vertices. The frontier vector is generated by comparing the result from current iteration with a list of activated vertices. The vertices

that are in the frontier are the vertices that were not activated in the previous iteration, but were activated in current iteration.

Our algorithm applies SpMV over the matrix and the frontier. For each iteration, each element in a row of the matrix is applied an AND (&&) operation with the corresponding element in the frontier vector. The results from these operations will be ORed (||) together, which gives the result for the row.:

```
    (martix_row[0] && frontier[0])||(matrix_row[1] && frontier[1])|| ...
2:  (matrix_row[n] && frontier[n])=result[row]
```

Unlike the breadth first search on boolean semiring, each node will have a chance to not be activated (set as '1'), even if matrix_row[x] and frontier[x] = 1. This resulted in that for each &&- operation, we need to && another *coin toss*, which determined if the activation takes place. The coin-toss is determined by the RNG and the global probability.

The algorithm will continue until either all of the vertices are activated, or no more vertices can be activated. This is solved with the dist_gen() function as we mention [INSERT PSEUDOCODE]. The function stops the algorithm when either no more vertices are activated in frontier vector, or all the vertices in the result vector is activated.

In the high level impleme the implementation of the the algorithm was done in two levels. The top level was the *TopLevelWrapper*. The TopLevelWrapper was set as the main function in our HLS implementation. The function takes in the address of the location of the matrix, the address of the result, and the address to the frontier. Since we are working with ICM, a global probability, a random seed as the initial state for the LFSR is also set as input. The TopLevelWrapper stores the useful data in a local buffer, where it is sent to the datapth-function. The datapath-function is a sparse martix vector multiplication.

The address to matrix, result,frontier, and the random seed and global probability is all mapped as AXI4-Lite while the matrix, frontier and the result are memory mapped. This allows us to send in the address of the memory location where the variables are stored and apply our algorithm.

Since ICM is dependent on a random function, we ran each sets of seed nodes 50 times to find the average runtime and coverage to find the most optimal set of nodes.

4.2 Linear-Feedback Shift Register

The LFSR is a commonly used pseudorandom number generator(pRNG)[39]. Different sized LFRS is able to generate a wider range of pseudo random numbers.

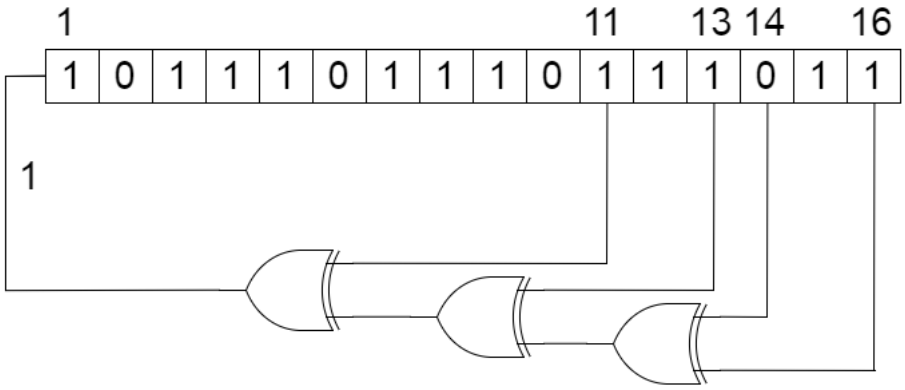


Figure 4.1: 16-bit Linear-Feedback Shift Register

LFSR generates a pseudorandom number based on the previous pseudorandom number it generated. The LFSR implemented for this project is a 16 bit shift register, that is able to generate a pseudorandom number in the range from 0-2^{16}(65536). In order to generate the new number, the bits from positions 16,14,13 and 11, is XORed. I.e. (((16 XOR 14) XOR 13) XOR 11) = new bit. The new bit is pushed into bit-position 1 and the entire registers shifts towards right 1 bit. This allows the IP-core to generate a pseudorandom number based on a initial seed input. From Figure: 4.1 we can see that to continuously generate pseudo random numbers the output from the LFSR is used as the next iterations seed.

4.3 Design flow of HLS

For this project, Xilinx Vivado HLS 2016.1 was used. The usual workflow in designing with Vivado HLS is as follow.

1. Define your funciton/algorithm.
2. Simulate as compile code.
3. Synthesise.
4. Co-simulate.

There are some steps that Vivado HLS requires before the project can start. In the begining, Vivado HLS would require the designer to specify which function is the top level of the implementation. That top function would determine which

port the IP core would have and what type of AXI4 protocol to implement. Vivado HLS also enables the user to specify to which platform this implementation is for.

The first step in designing with Vivado HLS is to define the algorithm that will be synthesised. E.g. for this report, its the matrix vector multiplication. After identifying different requirements and dependencies, the algorithm is implemented in C with Vivado HLS. Vivado HLS has some limitation regarding the high level implementations:

- No dynamic memory (Need to be static), Vivado HLS does not support malloc, free, new or delete.
- NO STD, FILE-IO, etc, (no system calls).
- avoid recursive functions.

The next step is to *run C simulation*. This will verify that the C implementation is correct, by running the test in the testbench. The test in the testbench are created by the designer. After verifying that the implementation is correct, next step is to synthesise the implementation. Vivado HLS will then generate the appropriate Verilog or VHDL, depending on the designer choice. The finished generated solution is then reviewed by reading the Vivado synthesise report. The report containing crucial information regarding the generated solution. There we can find the performance estimates of the generated core, the utilization estimates, and the interface to the generated core. After all this is done, Vivado offers a *run C/RTL Cosimulation*. Vivado will then run both the C simulation and testbench, and the same testbench on a simulated version of the implemented core. This function allows the designer to verify that the generated core have the same behaviour as the simulated C implementation.

After Cosimulation is done, the IP-core is ready for export. *Export RTL* generates the necessary RTL files and exports the IP core. The exported IP-core can then be found in the project folder and is ready to be uploaded to the FPGA.

The input and output port of the IP core is determined by the variables that the top function require. variables that are read from, will automatically be set as input, while variable that are only written to, will be set as output. In order for the core to understand what is mutable, the output is often set as a pointer (For C code). VIVADO HLS generates control signals cl automatically.

4.4 Network and graph generator

For this project, we choose to implement a R-mat generator as mentioned in Chapter: 2.8 and [1]. The generator is implemented in *Python* [40] with *numpy*.

The generator generates adjacency matrix with 2^k nodes, where k is known as the scale of the sparse matrix. The total amount of edges the graph contains is set as $total\_amount\_of\_edge = k \times \ edge\_factor$. The edge factor is the ratio between the graph's edge count and ins vertex count[41].

4.5 How is it connected

As shown in Figure 4.2, out IP-core is the *TopLevelWrapper*, and the PS is the Zynq processing system. The AXI interconnect serves as a interconnect between our IP-core and the PS.

Our core is memory mapped and we se the bus interface AXI4-Lite for communication. The core includes an dma-component, which the HLS would initiate for us. This allows us to utilize the High performance slave interface on our PS.

4.6 Xilinx SDK

Vivado offers a *Eclipse* based software develop kit (SDK). There we read the pregenerated graph from a SD card on the ZedBoard. We initiated the the custom IP core by setting all the required input signals.

4.7 Optimization

The function matrix_vector_multiplication() performes a single matrix vector multiplication. From the pseudocode, we can see that there is a room for parallization of the SpMV. The outer for loop from the pseudocode, can be parlized since the for loop is not dependent on the variables from the inner for-loop.

Another possible paralization is during the simulation, after the spmv, the frontier needs to be calculated. And a converged() function is called in the end to determen if the simulation is finished. The frontier calculation and converged can pe run in parallel.

include the different directives, PIPLINE

THe IP-core (currently) is only using around 2% of the resource available on the FPGA(Zedboard). This gives us a lot of room for parallization of different core. Implemented 2 bus, one for input stream, one for output stream. The output stream consist of the result_vector

4.8 Potential improvement

: For the second option, where the RNG core is not implemented into the IP CORE, we would have to have teh random number set as AXI4 stream from the buffer, since we would need to call a stream of random number from the core.

4.9 global vs local probability

For this project, a global activation chance was implemented, giving each node the same probability to be activated. A interesting model would be there are a personal probability of activation. Each node might have unique activation chance. Our implementation.

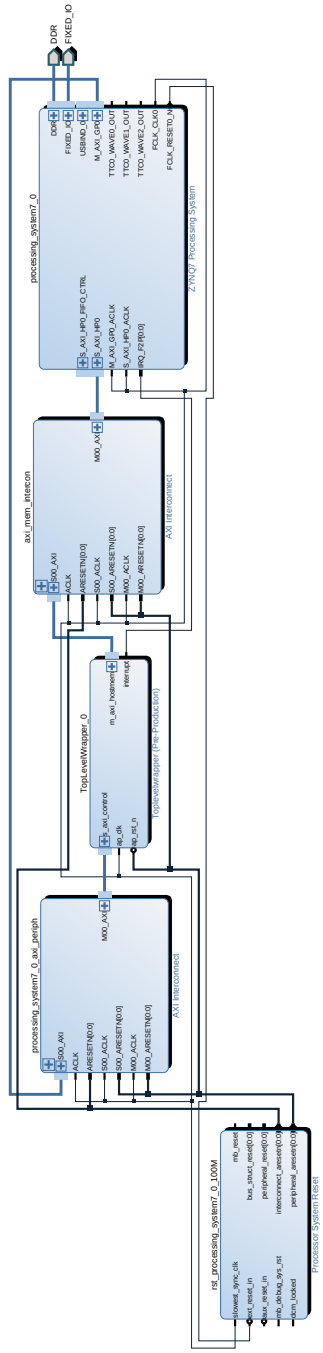


Figure 4.2: Our IP-core and how it is connected.

Chapter 5

Result and Discussion

as we can see, the algorithm was able to finish a We were able to run a implementation with pipeline and loop unrolling, but that resulted in a too large core and the synthesiser took over 24 hour to synthesise, Which was too long and was therefore abandoned. This resulted in that most of the numbers here are taken from Vivado HLS simulation and Cosimulation. —

5.1 Performance

As we can see, the hardware implementation is better then the C-simulated implementation, even at this low scale, we can see that the

We used the following variables to generate our adjacency matrix:

- **A** = 0.57
- **B** = 0.19
- **C** = 0.19
- **D** = 1-0.57-0.19-0.19 = 0.05
- **Edge factor** = 16
- **k (Scale)** = 10

By using the random-function provided by Python, we placed '1' in it's correct place. After the matrix is generated, we further applied a diagonal copying as shown in Figure:2.5. This is obligatory to generate an undirected graph.

For this project, a graph with size 1024 was used. This is a microscopic graph compared to graphs used in network analysis.

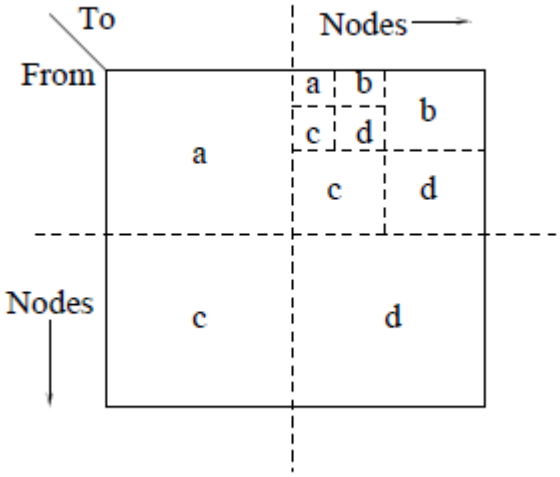


Figure 5.1: The R-mat model [1]

5.2 Discussion

Analysis of the performance

!

problem that was encountered

One problem that was encountered during this project was that the output signal from the synthesiser was not the correct direction. The output signal was often set as input signal. The HLS would automatically set the values as output signal or input signal.The reurn value from a funciton would be set as a the output signal, while the variable that the function takes, would be set as the input signal. Another way wo specify that something is the output signal would be to explicitly set them as pointer arguments. This will in set the signal to be output signal.

Another problem that I often encountered, was that the Vivado HLS often stop working. The problem was fixed by creating a new project and include the previous files.

Another problem, was that some times the Vivado HLS was not able to Cosimulate the implementation the first time. I was not able to find a solution to this

problem except resynthesis the project.

There were incidence where the implementation on the Zedboard did not behave as the implementation. This was solved by reprogramming the device and sometimes, restarting the Zedboard.

Chapter 6

Future work

Information diffusion and seed selection in general computes gigantic graphs, and thus is very time consuming. There is therefore several performance related improvements that have yet to be explored.

1. **Different architecture** for this implementation, we used a core including the LFSR, it would be interesting to explore different architecture. One solution that we did not have the chance to explore, is implement a large buffer connected to a single LFSR that continuously generates random number. The implemented cores would then each pop one random number for each cointoss. This solution requires a large buffer and would potentially generate large overhead with reading from the buffer. A large enough buffer would also be required since there are in worst case scenario for a single SpMV run, we would need n^2 cointoss. The potentially benefits of such a design would be better space utilization. A smaller core would use less resource of the Zedboard. This can result in more parallelization.

2. **Use larger graph** In graph theory, graph used is often at scale(26-42). The smallest mentioned graph from Graph500, is 2^26, a toy graph. Our graph is not even close to such a large graph.

3. **Customize algorithm** As mentioned in Chapter 3, HLS can generate a close to hand written design if the algorithm is customized for HLS. A interesting potential improvement would be to analyse algorithm and explore different solutions and implementation.

4. **Compare different solutions** In this report, we have only showed the result from one architecture, it would be interesting to compare different shcems

5. **Memory Optimization**. For this algorithm, we store the entire adacency matrix. This is inefficient for a sparse matrix. An potentially improvement would be to explore a different storage format for the adjacency matrix.

6. **Try different seed selection algorithm**

The community of the HLS is very active and frequently responds to forum post seeking help. Not many work that uses HLS[CITATION NEEDED]. Recently was free, used to cost money.

Learn more about HLS so it can be better utilized. different scheme to further work: - implement parallel - different scheeme, rng on the outside - Use other data structure. - larger graph - compare this solution to other solution, - implement more efficient memory storage - use other storage method since its sparse. - would be interesting to gather information on energy consumption. - Implement a more general architecture to handle more total nodes. - If there would be enough memory on the board, would be interesting to run 50 times on the board and just return the avrage time and coverage

Chapter 7

Conclusion

For placeholder, need to have some text here.

This paper works as a proof of concept, where we can see that with minimal knowledge regarding HLS, I was able to generate a simple Sparse Matrix vector multiplication IP-core. We can see from the result from co-simulation, that custom core ran much better then the other. HLS was easy to use and provided with a quick development time. There are still some problem with the HLS tools, but compared to earlier version, this is much better.

- HLS was great tool
- The custom core appears to be good.
- result is promising, shows potentiall
- HLS is great, allows rapid development and optimization.
- There is room for improvement, but still a promising field.
- There are still some random bugs in HLS.
- Not many clear tutorials for HLS

Bibliography

[1] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. 4:442–446, 2004.

[2] Daniel Gruhl, R. Guha, David Liben-Nowell, and Andrew Tomkins. Information diffusion through blogspace. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 491–501, New York, NY, USA, 2004. ACM.

[3] Daniel M. Romero, Brendan Meeder, and Jon Kleinberg. Differences in the mechanics of information diffusion across topics: Idioms, political hashtags, and complex contagion on twitter. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 695–704, New York, NY, USA, 2011. ACM.

[4] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and P Krishna Gummadi. Measuring user influence in twitter: The million follower fallacy. *ICWSM*, 10(10-17):30, 2010.

[5] J. Yang and J. Leskovec. Modeling information diffusion in implicit networks. In *2010 IEEE International Conference on Data Mining*, pages 599–608, Dec 2010.

[6] Seth A. Myers, Chenguang Zhu, and Jure Leskovec. Information diffusion and external influence in networks. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 33–41, New York, NY, USA, 2012. ACM.

[7] David Kempe, Jon Kleinberg, and va Tardos. Influential nodes in a diffusion model for social networks. 3580:1127–1138, 2005.

[8] M. C. McFarland, A. C. Parker, and R. Camposano. The high-level synthesis of digital systems. *Proceedings of the IEEE*, 78(2):301–318, Feb 1990.

[9] Donald Thomas and Philip Moorby. *The Verilog® Hardware Description Language*. Springer Science & Business Media, 2008.

[10] Y. Umuroglu, D. Morrison, and M. Jahre. Hybrid breadth-first search on a single-chip fpga-cpu heterogeneous platform. pages 1–8, Sept 2015.

[11] Éva Tardos David Kampe, Jon Klein. Maximizing the spread of influence through a social network. pages 137–146, 2003.

[12] Jeremy Kepner and John Gilbert. *Graph algorithms in the language of linear algebra*, volume 22. SIAM, 2011.

[13] MH McAndrew. On the product of directed graphs. *Proceedings of the American Mathematical Society*, 14(4):600–606, 1963.

[14] Pedro Domingos and Matt Richardson. Mining the network value of customers. pages 57–66, 2001.

[15] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 199–208, New York, NY, USA, 2009. ACM.

[16] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang. High-level synthesis for fpgas: From prototyping to deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, April 2011.

[17] Ritchie Zhao, Shreesha Srinath Gai Liu, Christopher Batten, and Zhiru Zhang. Improving high-level synthesis with decoupled data structure optimization. In *Proceedings of the 53rd Annual Design Automation Conference*, page 137. ACM, 2016.

[18] Wei Zuo, Yun Liang, Peng Li, Kyle Rupnow, Deming Chen, and Jason Cong. Improving high level synthesis optimization opportunity through polyhedral transformations. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, pages 9–18, New York, NY, USA, 2013. ACM.

[19] XIlinx. Zynq-7000 All Programmable SoC release notes, installation,and licensing,ug585 (v1.10), 2015.

[20] Xilinx. Zynq-7000 All Programmable SoC Overview, 2014.

[21] Kate Starbird and Leysia Palen. (how) will the revolution be retweeted?: Information diffusion and the 2011 egyptian uprising. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 7–16, New York, NY, USA, 2012. ACM.

[22] Eytan Adar and Lada A. Adamic. Tracking information epidemics in blogspace. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '05, pages 207–214, Washington, DC, USA, 2005. IEEE Computer Society.

[23] Manuel Gomez Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 1019–1028, New York, NY, USA, 2010. ACM.

[24] Eytan Bakshy, Jake M. Hofman, Winter A. Mason, and Duncan J. Watts. Everyone's an influencer: Quantifying influence on twitter. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 65–74, New York, NY, USA, 2011. ACM.

[25] Grant Martin and Gary Smith. High-level synthesis: Past, present, and future. *IEEE Design &amp Test of Computers*, 26(4):18–25, 2009.

[26] S. Director, A. Parker, D. Siewiorek, and D. Thomas. A design methodology and computer aids for digital vlsi systems. *IEEE Transactions on Circuits and Systems*, 28(7):634–645, Jul 1981.

[27] A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive, and J. Kim. The cmu design automation system: An example of automated data path design. In *Proceedings of the 16th Design Automation Conference*, DAC '79, pages 73–80, Piscataway, NJ, USA, 1979. IEEE Press.

[28] John Granacki, David Knapp, and Alice Parker. The adam advanced design automation system: Overview, planner and natural language interface. In *Proceedings of the 22Nd ACM/IEEE Design Automation Conference*, DAC '85, pages 727–730, Piscataway, NJ, USA, 1985. IEEE Press.

[29] P. G. Paulin, J. P. Knight, and E. F. Girczyc. Hal: A multi-paradigm approach to automatic data path synthesis. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, DAC '86, pages 263–270, Piscataway, NJ, USA, 1986. IEEE Press.

[30] H. D. Man, J. Rabaey, P. Six, and L. Claesen. Cathedral-ii: A silicon compiler for digital signal processing. *IEEE Design Test of Computers*, 3(6):13–25, Dec 1986.

[31] D. Navarro, Luca, L. A. Barragn, I. Urriza, and Jimnez. High-level synthesis for accelerating the fpga implementation of computationally demanding control algorithms for power converters. *IEEE Transactions on Industrial Informatics*, 9(3):1371–1379, Aug 2013.

[32] XIlinx. Vivado Design Suite User Guide release notes, installation, and licensing,ug973 (v2015.4), 2015.

[33] F. Winterstein, S. Bayliss, and G. A. Constantinides. High-level synthesis of dynamic data structures: A case study using vivado hls. In *Field-Programmable Technology (FPT), 2013 International Conference on*, pages 362–365, Dec 2013.

[34] Gorker Alp Malazgirt, Nehir Sonmez, Arda Yurdakul, Adrian Cristal, and Osman Unsal. High level synthesis based hardware accelerator design for processing sql queries. In *Proceedings of the 12th FPGAworld Conference 2015*, FPGAworld '15, pages 27–32, New York, NY, USA, 2015. ACM.

[35] Donald G. Bailey. The advantages and limitations of high level synthesis for fpga based image processing. In *Proceedings of the 9th International Conference on Distributed Smart Cameras*, ICDSC '15, pages 134–139, New York, NY, USA, 2015. ACM.

[36] Shahzad Ahmad Butt, Mehdi Roozmeh, and Luciano Lavagno. Designing parameterizable hardware ips in a model-based design environment for high-level synthesis. *ACM Trans. Embed. Comput. Syst.*, 15(2):32:1–32:28, February 2016.

[37] Scott Beamer, Krste Asanović, and David Patterson. Direction-optimizing breadth-first search. *Scientific Programming*, 21(3-4):137–148, 2013.

[38] Virat Agarwal, Fabrizio Petrini, Davide Pasetto, and David A. Bader. Scalable graph exploration on multicore processors. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[39] M. Murase. Linear feedback shift register, February 18 1992. US Patent 5,090,035.

[40] Anaconda. Anaconda python 2.7, 2015.

[41] Graph 500 Steering Committee. Graph 500 Benchmark 1 ("search"), 2010.