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 vertices, known as *seed nodes*. These seed nodes pass 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 vertices, we can find important targets for viral marketing by giving free samples, and use this information to spread messages quickly during disaster scenarios[2] [3].

There are multiple studies done regarding information diffusion, [2],[4], [5], [6]. But as far as we know, there are none that focuses on optimizing the seed selection in hardware. The current seed selection algorithm is a greedy solution[7], where every set of vertices is tested and the set with the 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 constraints 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 semiring. 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}) vertices)

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 networks, 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 generated 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 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.

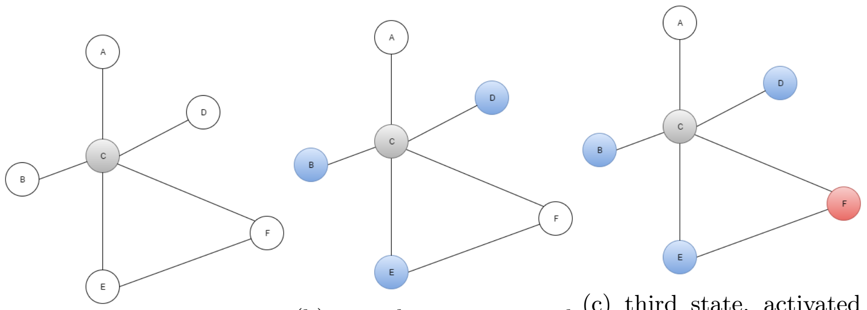
2.1 Information Diffusion

Information diffusion is looking at how information is propagated through a network. A vertex can be either activated(infected) or inactivated(healthy/noninfected), each vertex 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 consists of a set of starter vertices, which we will call seed nodes, which are "infected" at initial time step. During each time step, there are a percentage p_g where the "infected" vertices would "infect" its neighbours. Seed nodes is a set of k vertices that in the initial time-step are infected. They will pass on the information/infection during each iteration, 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

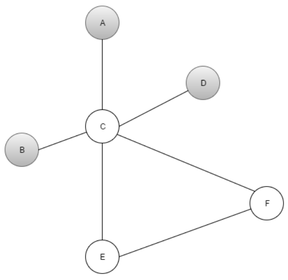
(b) second state, activated by C is marked blue

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

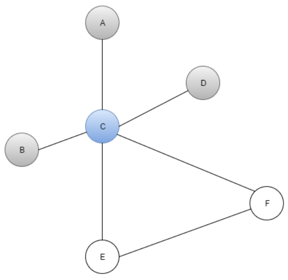
Figure 2.1: Independent cascade model

For the LTM activation is not dependent on a probability and a ”coin flip”, but an 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 than 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.

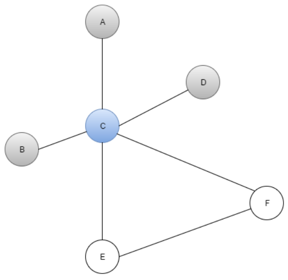
A real world example of the LTM would be e.g. A new product on the marked. People would adopt the new product it enough of their acquaintance is



(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 vertex v_r. The algorithm then stores all v_rs children vertices in a *queue*. The algorithm then takes the first vertex from the queue, v_1 and stores all the children vertices to v_1 in the back of the queue. This process continues until the queue is empty and all the vertices have been iterated over.

BFS is a common graph iteration algorithm but is often limited by the irregular memory access where the algorithm has 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: $dist[\forall v \in V] = -1; currentQ, nextQ = \emptyset$
2: $step = 0; dist[root] = step$
3: ENQUEUE(nextQ,root)
4: while $nextQ \neq \emptyset$ do
5:     $currentQ = newxtQ; nextQ = \emptyset$
6:     $step = step + 1$
7:     while $currentQ \neq \emptyset$ do
8:         $u$ = DEQUEUE(currentQ)
9:         for $v \in Adj[u]$ do
10:            if $dist[v] == -1$ then
11:                $dist[v] = step$
12:                ENQUEUE(nextQ, v)
   return dist
```

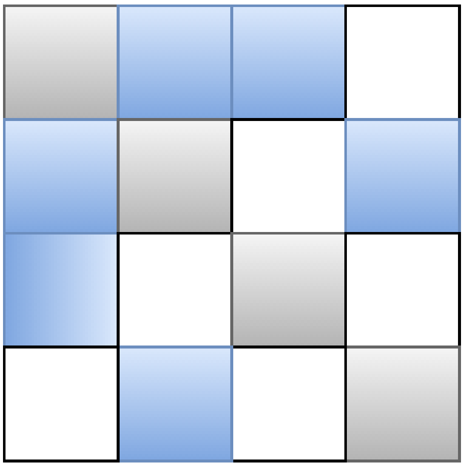
matrix-vector multiplication.

2.4 Matrix Notations

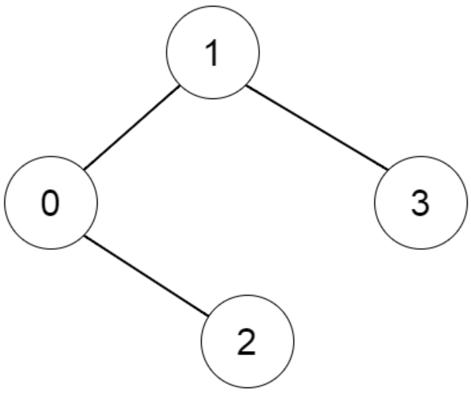
vertices and edges are not the only way to represent a network. Networks can be represented as *sparse adjacency matrices* [12] [13]. By representing networks as a sparse matrix, we can often discover different ways to optimize the algorithm, or a different structure to store the data. The adjacency matrix, in particular, is an interesting way to represent the graph. A graph $G = (V, E)$, G have N vertices and M edges, and 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, it is 0. In Figure:**??**, we can see how a undirected graph can be represented as an adjacency matrix. Each square symbolises a connection between two vertices. To generate a undirected graph as an 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. A social graph with few edges would often be represented as a sparse matrix. Since sparse matrices 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 an adjacency matrix A, then for the root vertex, a vector x(root)=1 is multiplied with the matrix A. $A \times x_0 = y_0$. y_0 is the result of 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 vertices is in the initial timestep activated. During each time step, the seed nodes

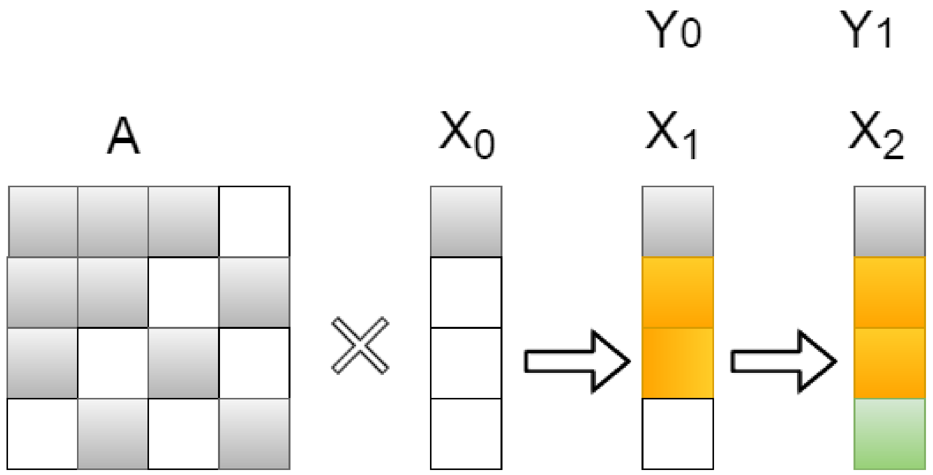


Figure 2.4: BFS on Boolean semiring

will 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 schemes 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 iterate over the entire network and finds one vertex that has the largest coverage and stores that vertex in the set S. The algorithm then activates that vertex from A, and computes the coverage of each vertex in the network with the previously chosen vertex. After the algorithm finds the second vertex, that vertex is stored in A with the previous vertex and we have now chosen two seed vertex. The algorithm continues until we have chosen k seed nodes, where each has been tested to have maximum coverage in relation with the previously chosen vertices. To compute the maximum coverage, the algorithm has to test every combination of vertices 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 transition probability. Each vertex 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 vertex $x$, use repeated sampling to approximate $\sigma(A \cup x)$ to
       within $(1 \pm \varepsilon)$ with probability $1\delta$
4:     Add the vertex with largest estimate for $\sigma(A \cup x)$ to A.
5: Output the set $A$ of vertices.
```

2.5.2 The Degree Algorithm

Another popular algorithm is the degree algorithm [7]. Unlike the greedy algorithm, does not compute the coverage of vertex, the algorithm picks the top k vertices according to the degree distribution instead. The vertex chooses the top k vertices 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 iteration is needed to compute the degree to the vertex. The disadvantage is that this algorithm does not take the degree correlation into account. As mentioned in section **??**, high degree vertices would often have common vertex as neighbours. This would result in multiple overlapping activated vertices chosen.

Algorithm 3 Degree Algorithm

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

2.5.3 Independent Algorithm

Another algorithm is the independent greedy algorithm. The algorithm iterates through the network, computing the spread of each vertex. The algorithm then chooses the vertex with the largest coverage independent of the other previous chosen vertices. 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 vertex $x$, use repeated sampling to approximate $\sigma(A \cup x)$ to
       within $(1 \pm \varepsilon)$ with probability $1\delta$
4:     Add the vertex with largest estimate for $\sigma(x)$ to A.
5: Output the set $A$ of vertices.
```

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 convert algorithms implemented on higher level down to *Register Transfer Level* (RTL)[16]. RTL is models of digital circuits displaying the flow of data between register, logical operations and such. It is commonly used to describe 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 languages 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 has 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 are 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 networks with only a few parameters, the generated graph will naturally have the small world properties and follows the laws of normal vertices, and have a quick generation speed [1]. The R-mat models goal is to generate graphs that match 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 is capable of generating graphs with total numbers of vertex $V = 2^x$. 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.

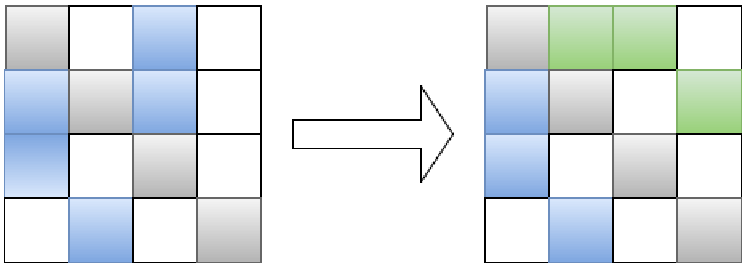


Figure 2.5: How the adjacency matrix is flipped on the diagonal

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 than 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 vertices, but also an external influence. They found that a 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 study 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 vertex is either *activated*(infected, influenced, '1') or *inactive*(healthy, not reached, '0'), and passes the *contagion*(information, data, infection, influence) to neighbouring vertices 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 of the network is sufficient to explain the behaviour[5]. The report proposes an alternative model, *Linear Influence Model*(LIM), where the focus is on the global influence that an infected vertex has on the rate of diffusion through the implicit network. This model makes the assumption that newly activated vertices are dependent on previously activated vertices. The LIM does not need explicit knowledge of the entire network. Instead, the model takes the newly activated vertices 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 has 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. Many HLS tools were built in later years mostly for prototyping and research[28][29][30]. Some of these were able to produce real chips, but the reason for the lack of further development and adaptation was that RTL synthesis was not a widely accepted and an immature field. This often lead to suboptimal solutions.

Around the year 2000, new HLS tools were developed in academia and the industry. These tools, used high 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 a 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 discusses 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", and "Lack of a compelling Reason/event to adopt new design methodology". Early versions of HLS tools was not a C-to-RTL transformation. Most of them needed a custom *Hardware Descriptive Language*(HDL). Lacking reusable and portable design specification resulted in that HLS tools required users to include detailed information regarding timing and interface information into source codes. 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 an interface to other hardware modules and platform integration. Those aspects were left to the users to solve system integration problem. The lack of foundation to accurately measure HLS result and

often failed to meet timing and power requirement with early HLS tools were another limiting factor. The last reason was that there was no real driving force to turn developers over to such a young and early development format. HLS tools showed exciting 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 tremendous 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 a focus on accelerating operations; 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 accelerators 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 advantages and disadvantages 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 an 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 a replacement for hardware designers or clever designs.

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

[18] discuss improvements to the current HLS tools with polyhedral transformation. Here they present 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 patterns, 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 are applied. Then an 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 were 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 customize 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 the breadth-first search. By performing it as a sparse matrix-vector multiplication over a boolean semiring, more parallelization 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 vertices on that level, before each vertex checks its neighbour for unvisited vertices. Unvisited vertices are placed in the frontier vector and marked as visited. The bottom-up algorithm, contrary to the top -down the algorithm, is where each

children vertex tries to find a potential parent. A neighbour vertex can be the parent if the neighbour is also in the frontier vector. 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

Foreword

Chapter 5

Introduction

5.1 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 vertices, known as *seed nodes.* These seed nodes pass 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 vertices, we can find important targets for viral marketing by giving free samples, and use this information to spread messages quickly during disaster scenarios[2] [3].

There are multiple studies done regarding information diffusion, [2],[4], [5], [6]. But as far as we know, there are none that focuses on optimizing the seed selection in hardware. The current seed selection algorithm is a greedy solution[7], where every set of vertices is tested and the set with the 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 constraints to lower level design.[8]. It makes it possible to implement an algorithm in a 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 semiring. This is done by using HLS as the development tool.

5.2 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}) vertices)

5.3 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 networks, 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 generated 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 concluding remarks regarding this paper and a summary of the identified tasks.

Chapter 6

Result and Discussion

Here we will present the results that we received from the experiments. The algorithm was able to finish an extreme scaled down version of the original sparse matrix-vector multiplication. The result was a 16×16 adjacency matrix, which is unfortunate since our goal was to iterate over large graphs. But these results are still interesting and proves that HLS can be an improvement over traditional hardware design.

We were able to run an implementation that utilizes pipeline and loop unrolling, but that resulted in a too large core, and the synthesiser took over 24 hours 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.

6.1 Results

In Table 6.2, we can see that by varying the buffer size, we get variable results. We can see that in C simulation, by using a larger buffer so that the IP core can take the entire row of the matrix, the result is somewhat better than the other. While for the RTL implementation, there does not seem to be the case. For the RTL simulation, by choosing buffer size equal to eight, get a better time.

from these results, we can clearly see that the RTL simulation is faster then the C simulation.

Table 6.1: Results from Zedboard, different sized seed nodes.

k	coverage	Time usage(difusin)in microseconds	Seed selection time usage in micro sec
1	0.198750	32.74 us	0.45 us
2	0.247500	936.88 us	0.87 us
3	0.306250	1237.62 us	1.26 us
4	0.391250	1501.96 us	1.63 us
5	0.460000	1936.74 us	1.97 us

Table 6.2: Results from simulation with vary buffer size

Buffersize	Time in milliseconds (with C simulation)	Time in milliseconds (with RTL sim
4	7.200 ms	0.920 ms
8	6.260 ms	0.680 ms
16	5.180 ms	0.760 ms

6.2 Performance

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

We have here included a snapshot of our synthesis report. As we can see, we have a rather large usage of look-up table(LUT) and flip-flop(FF) for our design. This is the result of the PIPELINE and Loop unrolling. Our high-level implementation does contain several Loops and dependencies; this results in a significant amount of resource used.

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)** = 4

By using the random() function provided by Python, we placed '1' in its designated position. After the matrix was generated, we further applied a diagonal copying as shown in Figure:2.5. This is obligatory to create an undirected graph.

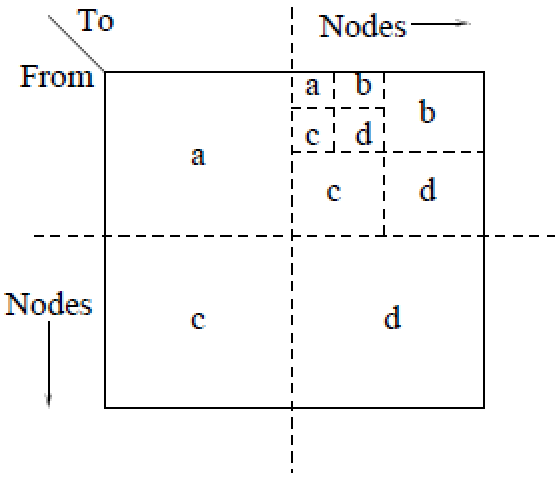


Figure 6.1: The R-mat model [1]

6.3 Discussion

Analysis of the performance

Our IP core was originally designed for networks with 1024 vertices, but after implementing with the optimization directives, the resource usage was out of bound. This resulted in a severe scale down for this project. The resulted network was down to the size of 16 vertices. In network analysis, this is such a small scale that the results from this experiment would only serve as a proof of concepts.

The original idea with table 6.2 was to observe how the IP-core behaved when it would receive part of the matrix row. By finding an optimal buffer size that would satiate the bandwidth and still receive enough elements to compute. The result we got was not very representative. Since our implementation and the matrix we used was so small compared to other networks, we would not have been close to satiate the bandwidth.

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 return value from a function would be set as an output signal, while the variable that the function takes, would be set as the input signal. Another way to specify that something is the output signal would be to explicitly set them as pointer arguments. This will in set the signal to be the 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 sometimes the Vivado HLS was not able to Cosimulate the implementation the first time. I was not able to find a solution to this issue except resynthesis the project.

There was 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 7

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 vertices. - 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 8

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.