Individual Diploma Thesis

Speeding up the process of generating stress tests

PANTELEIMONAS CHATZIMILTIS

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2023

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

Speeding up the process of generating stress tests

PANTELEIMONAS CHATZIMILTIS

Supervisor

Yiannakis Sazeides

The Individual Diploma Thesis was submitted for partial fulfilment of the requirements for obtaining a degree in Informatics of the Department of Informatics of the University of Cyprus

May 2023

Acknowledgments

I would like to thank all the following people who made this work possible.

Foremost, I would like to thank my supervisor Dr. Yanos Sazeides, Professor at the Department of Computer Science at the University of Cyprus, for his guidance and support throughout the research process. His vast knowledge, valuable insights, and constructive feedback helped me a lot in developing my ideas and refining my research approach.

I am also grateful to Georgia Antoniou for her constant support, helpful discussions, and valuable guidance throughout the entirety of the research process. Her guidance was very helpful and had a huge impact on the outcome of this work.

Moreover, I would like to thank Dr. Jawad Haj-Yahya for his invaluable insights and helpful feedback that contributed significantly to the success of this work. His expertise in processor design and architecture, combined with his willingness to share his knowledge, made significant impact on my research.

Finally, I am deeply grateful to my friends for their encouragement and support throughout this journey. Without them this work wouldn’t be possible.

Abstract

In this thesis, we address the problem of optimizing stress-test generation and enhancing power consumption in computer systems. The execution time of stress-test generation is a critical factor in system testing, and traditional methods often lack efficiency. Moreover, maximizing power consumption in stress-test individuals can provide valuable insights into system behavior. To tackle these challenges, we propose a novel approach called GeST-R, which integrates Generic Algorithms (GAs) with Euclidean Distance and Regression Models.

In our research, we investigate the accuracy of regression models in predicting the characteristics of the “best-virus” after some generations. By analyzing the relationship between system characteristics and performance metrics, these models provide valuable insights into stress-test optimization. Additionally, we explore the effectiveness of Euclidean Distance as an alternative fitness metric, aiming to replace power consumption in the GA framework, since it requires a significant amount of time.

Overall, the goal of our thesis is to optimize stress-test generation using GeST-R, a new approach that combines GAs, Euclidean Distance, and Regression Models. We aim to improve the efficiency and effectiveness of stress-test generation by accurately predicting characteristics and enhancing power consumption. Our research contributes to the advancement of computer system testing methodologies, providing valuable insights for future investigations in the field.

Table of Contents

Chapter 1

Introduction

1.1 Problem definition

1.2 Research questions

1.3 Contributions

1.4 Thesis Structure

1.1 Problem definition

Stress-tests are an important tool for system reliability testing and performance enhancement. They are small programs that stress different aspects of a computing system such as power consumption or voltage noise, with the aim to detect any architectural, PDN vulnerabilities or performance bottlenecks.

The creation of these viruses can be accomplished through either manual or automatic methods. However, it is essential to have a deep understanding of the target device's architecture to maximize the microarchitectural activity effectively. This is particularly important for power viruses [2], where maximizing the processor's power consumption is critical.

The main problem with both of these approaches is their time-consuming nature. Manually writing a stress-test can be a challenging, trial-and-error procedure that requires a significant investment of time. Additionally, current automatic methodologies based on genetic algorithms (e.g., GeST framework [1]), require the compilation and execution of each stress test to obtain its power consumption, which is also time-consuming.

For these reasons, a new method QuickGen [3] has been introduced to address the GeST's time-consuming problem. This approach involves optimizing the framework to bypass the compilation and execution of each program. However, the issue of this approach is that it relies on an existing good reference virus and uses the Euclidean distance (the difference on number and type of instructions) of that reference virus from the candidate viruses in order to drive the

optimization search. Unfortunately, the reference virus may not always be available. The Euclidean distance does not cover the whole design space of the problem (it only considers only the instruction types).

1.2 Research questions

The following are the research questions answered in this thesis:

1. Can the execution time of the stress-test generation be optimized using Genetic Algorithms with the inclusion of Euclidean Distance (QuickGen) and Regression Models?
    a) Can the regression model accurately predict the characteristics of the "best-virus" after X number of generations?
    b) Can Euclidean Distance metric effectively replace Power Consumption as the fitness value of the Genetic Algorithm?
2. Is there a method to enhance the power consumption of GeST individuals (stress-tests), specifically targeting Sandy Bridge processors?
    a) Can aligning the stress-test address of the GeST individual improve data storage in cache blocks and consequently result in higher power consumption?
    b) Does stack initialization contribute to a higher power consumption?

1.3 Contributions

In my research, I aim to make several significant contributions to the problem.

- Development and validation of a methodology that predicts the characteristics of a good power virus after a certain number of generations (X).
- Implementation of the methodology in GeST, an existing tool for automatic stress test generation, along with the integration of the QuickGen methodology.
- Experimental evaluation of the methodology, demonstrating that GeST-R can generate better power viruses in a faster manner.
- Investigation of the impact of aligning the loop address of GeST individuals on power consumption. Although the results show slight improvement, further exploration is needed to fully understand its effects.
- Finding that stack initialization does not significantly contribute to an increase in power consumption. These finding offer valuable insights, in the context of Sandy Bridge processors.

1.4 Thesis Structure

The remaining structure of the thesis is as follows:

Chapter 2 and Chapter 3 provide the necessary background information, machine I used, processor, and instruction set architecture (ISA). It also explains essential metrics such as power consumption and Euclidean distance, as well as regression models, GeST and QuickGen methodologies.

Chapter 4 describes the proposed method, specifically GeST-R methodology, designed to improve stress-test generation time.

Chapter 5 focuses to the experimental setup used to evaluate the proposed method. It describes the hardware configuration, the methods used for evaluation, and the validation effort.

Chapter 6 presents the results obtained from the experimental evaluation of the proposed method. It analyzes and discusses the findings, highlighting the effectiveness and efficiency of the GeST-R approach.

Chapter 7 examines the related work, discussing another research and approach to stress-test generation.

Chapter 8 looks to the future, discussing potential paths for further research and development in the field, including possible extensions or improvements to the GeST-R method.

Finally, Chapter 9 concludes the thesis by summarizing the main contributions.

Chapter 2

Background

2.1 Machine & Processor & ISA

This research focuses on developing a program optimized for specific machine, making it crucial to understand the machine components. Each computer has its unique characteristics, and the program's performance, particularly in terms of power consumption, varies across different machines. Therefore, a comprehensive understanding of the machine being used is essential to achieve maximum program efficiency.

The machine used in this research for testing is the HP Compaq 8200 Elite CMT PC, equipped with an Inter® Core™ i5-2400 CPU @3.10GHz. To conduct experiments, we will utilize two instances, meaning only two of the cores will be utilized.

The processor has a 6MB cache and a bus speed of 5 GT/s. Examining the microarchitecture of the processor, we find that the L1 instruction cache can accept 16 bytes per cycle. Additionally, the processor has a 4-way decode, allowing it to fetch four instructions simultaneously. The architecture is made out of tree main parts, frontend, execution engines and memory. Frontend is responsible to fetch and decode instructions. Because Intel is RISC based architecture it decodes the CISC ISA to RISC (micro-Ops) dynamically in the fetch part of the processor. Also, there is a micro-Op cache. The microarchitecture also comprises six execution units, enabling the execution of six instructions concurrently. The execution ports are organized as follows:

- The **first** unit supports the integer arithmetic logic unit, integer division, integer vector arithmetic logic unit, integer vector multiply, floating-point multiply, floating-point division and vector shuffle.
- The **second** unit supports the arithmetic logic unit, integer multiply, integer vector arithmetic logic unit, and floating-point add.
- The **third** unit supports the integer arithmetic logic unit, vector shuffle, integer vector arithmetic logic unit, and branch.

- The **fourth** and **fifth** units support address generation unit (AGU) and load data operations.
- The **sixth** unit supports store data operations.

Based on this information, we can observe that three units focus on computation, while the remaining three units focus on memory access. Regarding computation, all three units support integer and integer vector arithmetic logic units, with only two units supporting floating-point computation and one unit supporting branches. Among the units supporting memory access, two units handle load operations and AGU while one unit manages store operations.

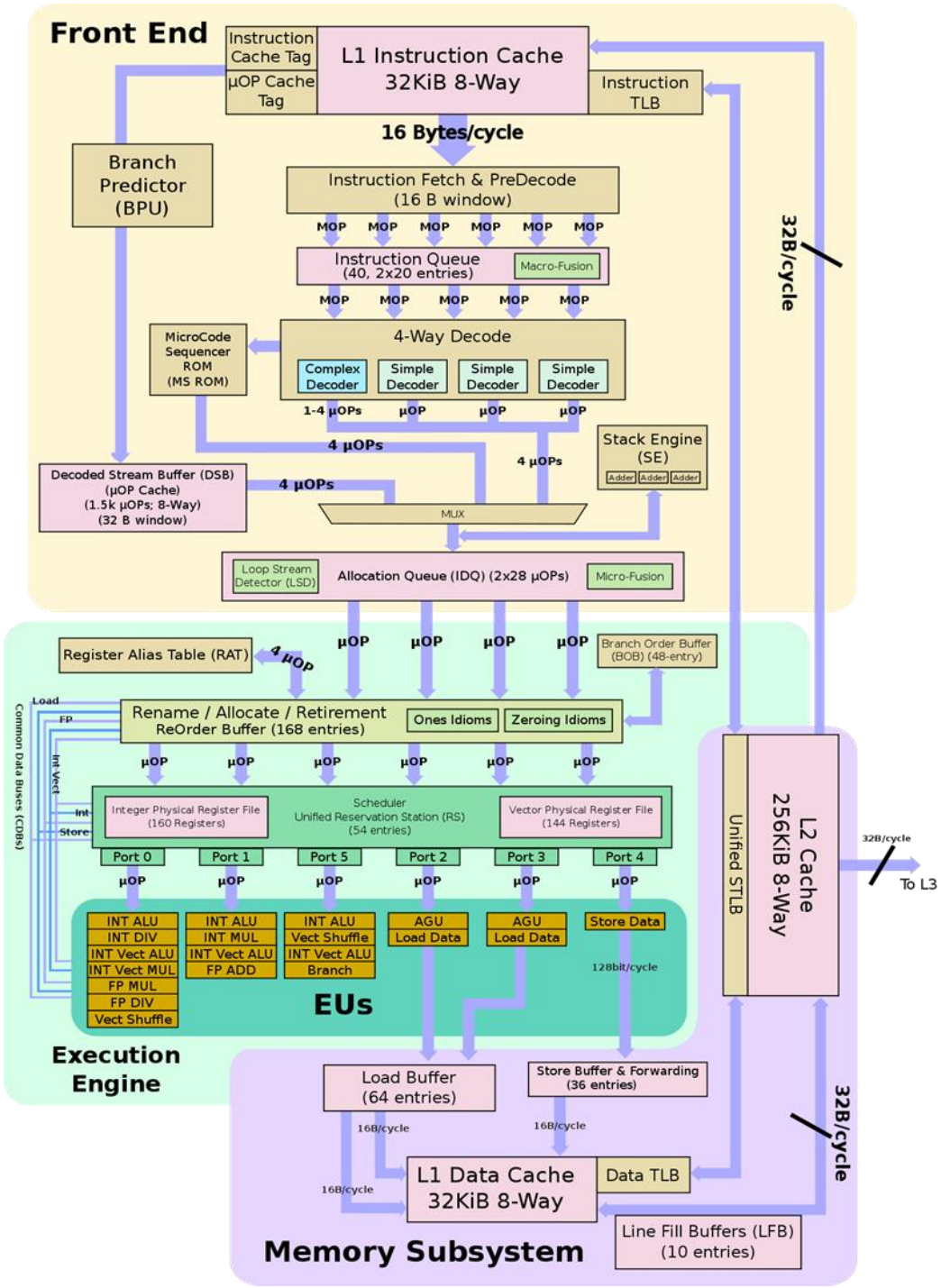

Figure 2.1.1 Microarchitecture of Sandy-Bridge processor

Figure 2.1.1, illustrates the architecture that the research focuses on. The x86 ISA defines how the processor processes and executes various instructions issued by the operating system (OS) and software programs. It serves as a logical framework for instruction execution and enables programs and instructions to run on any processor belonging to the Intel 8086 family.

The x86 architecture encompasses key characteristics that facilitate efficient instruction execution. Firstly, it provides a framework for the processor to interpret and execute instructions consistently. Additionally, it provides a consistent execution environment for programs and instructions across various processors within the Intel 8086 family.

Furthermore, the x86 architecture incorporates mechanisms for utilizing and managing the CPU's hardware components. This includes functionalities related to memory addressing, software and hardware interrupt handling, data types, registers and input/output management. The architecture plays a vital role in supporting program functionality and offering services necessary for efficient computation and system operation.

In this research, the focus will be on studying the instructions of the instructions of the x86 architecture. The following types of instructions will be explored:

- Scalar instructions: These instructions perform computations on a single number or a set of data at a time.
- Vector instructions: Multiple operations of the same type are carried out on several datasets simultaneously within a single processor instruction.
- Load instructions: These instructions are used to transfer data or memory addresses from memory to a register.
- Store instructions: These instructions move the value stored in a register to memory.

Additionally, the research will examine combinations of different instruction types.

Dependencies between instructions are particular interest, as they introduce latencies in program execution. The following basic dependencies can exist:

- Read after Write (RAW) dependence: This dependence exists when instruction B attempts to read data before instruction A writes it.
- Write after Write (WAW) dependence: This dependence exists when instruction A tries to write output before instruction B writes it.

- Write after Read (WAR) dependence: This dependence exists when instruction B tries to write data before instruction A reads it.

WAW and WAR hazards can be overcome relatively easily by implementing register renaming through software of hardware. However, RAW hazards pose a greater challenge, as they cannot be easily resolved. One approach to mitigating RAW dependencies is through the implementation of out-of-order execution. Other mitigation approaches include, stall execution, forwarding and value prediction.

```
vmulpd %ymm8,%ymm10,%ymm8
mov 0(%rsp),%rax
mov 64(%rsp),%rdx
mov 128(%rsp),%rdi
shl $31,%rbx
vmaxpd %ymm15,%ymm9,%ymm13
vmulpd %ymm15,%ymm5,%ymm11
vmulpd %ymm5,%ymm8,%ymm9
sar $31,%rdx
add $1216907345,%rdi
vmulpd %ymm5,%ymm11,%ymm10
mov 0(%rsp),%rdi
mov 64(%rsp),%rax
mov 128(%rsp),%rdi
add %rdx,%rsi
vmulpd %ymm9,%ymm12,%ymm15
vaddpd %ymm5,%ymm6,%ymm11
add %rbx,60(%rsp)
imul %rbx,%rsi
vaddpd %ymm6,%ymm7,%ymm8
vmulpd %ymm8,%ymm8,%ymm6
vaddpd %ymm2,%ymm1,%ymm0
vmulpd %ymm11,%ymm1,%ymm7
mov %rdx,%rdx
vmulpd %ymm1,%ymm15,%ymm3
mov %rdi,100(%rsp)
vmulpd %ymm2,%ymm13,%ymm5
mov 84(%rsp),%rbx
```

Figure 2.1.2 Example of code of x86 architecture

Figure 2.1.2 provides an example of code written in x86 ISA, offering a visual representation to help identify different types of instructions. The code includes the following examples of instructions:

- Vector instructions: vmulpd, vaddpd, and vmaxpd
- Scalar instructions: add, sar, shl, mov

- Load instruction: mov 84(%rsp), %rbx
- Store instruction: mov %rdi, 100(%rsp)
- Combination of Scalar and store: add %rbx, 60(%rsp)

Analyzing this code example allows for a better understanding of the different instruction types present in the x86 ISA.

2.2 Power Consumption

Power consumption refers to the amount of energy consumed per unit of time, typically measured in watts. Each machine has an idle power consumption, which indicates the power consumed when the machine is not actively used by any program. However, the power consumption changes as the machine transitions from the idle state to running a process on the CPU. The power consumption increases based on the computational intensity of the program running on the CPU. Additionally, when multiple programs are running on multiple cores of the CPU, the power consumption is influenced by both the number of programs and their computational demands. It's important to note that higher power consumption also leads to increased system temperature.

Core C-states, also known as processor idle sleep states, are power-saving states in which the CPU disables or lowers the functioning of specific components. Each processor supports its own set of Core C-states, and higher Core C-states typically result in more components being turned off or reduced in operation. This helps to decrease power consumption by minimizing unnecessary energy usage. They save power when the system is idle, which is a necessary requirement. Also, they provide power performance trade-offs, the more components switched off the more power saving. Moreover, some C-states may be hidden from the operation system, providing additional power-saving capabilities.

P-states, or power performance states, are states in which the frequency and voltage of the CPU are adjusted to optimize power consumption. Each processor has a specific number of P-states that can be utilized to dynamically modify the CPU's operation parameters, such as frequency and voltage. By adjusting these parameters, the CPU can achieve a balance between performance and power consumption, aiming to reduce overall power consumption while maintaining desired performance levels. Also, usually P0 is the settings for turbo and P1 are the normal settings.

2.3 Euclidean Distance

Euclidean distance is a statistical method used to measure the dissimilarity between two sets of values. In this method, a list of reference values is compared with corresponding values from other lists. The deviation between the two lists is calculated by subtracting the corresponding values and squaring the result. Finally, the squared differences are summed, and the square root of the sum is taken.

The Euclidean distance formula is as follows:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

Figure 2.3.1 Euclidean distance formula

Where x and y represent vectors, by applying this formula, the Euclidean distance can provide insights into the dissimilarity or similarity between different sets of values.

2.4 Regression Models

A regression model provides a mathematical function that describes the relationship between one or more independent variables and a response, dependent, or target variable. It helps us understand how changes in the independent variables are associated with changes in the target variable.

For instance, consider the relationship between height and weight. This relationship can be described using a linear regression model, where height is the independent variable and weight is the dependent variable. By analyzing the data and fitting a regression line, we can estimate the weight based on the given height.

Regression analysis serves as the foundation for various prediction tasks and helps us determine the effects on target variables. In this thesis, regression models are used to predict the features of the “best” power-virus.

Additionally, it is worth noting that the regression models used in this research are logarithmic regression models. These models are specifically chosen due to the fact that they seem to “follow” the trends of the architectural feature limits for maximum power consumption. A unique characteristic of logarithmic regression is its potential to reach a saturation point, where subsequent increases in the input variables (such as Generation) do not lead to significant changes in the predicted features.

Chapter 3

Virus programs & GeST

3.1 Viruses

Computer viruses are malicious programs with the primary intention of causing harm. They can replicate themselves and spread across networks to infect other computers. The purposes of viruses can vary depending on their type, but in general, they have the ability to disrupt data, cause damage to the operating system, and result in data loss or leakage. Viruses typically operate by utilizing executable files that often hide behind running processes, making them challenging to trace.

3.2 Stress-Tests

Stress-testing is a crucial process for assessing the performance and reliability of various systems, including computers, networks, programs, and devices. Its objective is to subject these systems to extreme conditions and determine if they can maintain a satisfactory level of effectiveness. Stress-tests involve putting systems through demanding scenarios and observing their response.

During stress-testing, systems are evaluated individually on each aspect to identify any vulnerabilities or weaknesses that may arise. This comprehensive examination is crucial as vulnerabilities are often easier to detect under stress conditions. Although stress-testing can be time-consuming, it is a mandatory step in development of robust and functional system.

The primary goals of stress-testing are to determine if a system can effectively operate under normal circumstances and also that the system maintains limited functionality even when a large part of it has been compromised. By undergoing stress-tests, systems can be better prepared to handle unexpected situations, ensuring their reliability and performance in critical scenarios.

There are two methods for generating stress tests: manual and automatic. The manual approach involves the programmer manually writing and testing the stress-tests. On the other hand, the automatic approach relies on specialized tools that use optimization search algorithms, such as Genetic Algorithms (GeST), to generate the stress tests based on specified optimization metrics.

3.3 Power Virus

The power virus is a specific type of virus commonly utilized in stress tests to assess the resilience of a system under extreme conditions. Its main characteristic is its ability to induce maximum CPU power consumption by executing a machine code-based program. This high CPU power consumption leads to increased heat generation within the system. Without adequate cooling mechanisms in place, this excessive heat can result in data corruption, faulty calculations, and system crashes. In some cases, it may even cause permanent damage to the system.

While power viruses are primarily employed in stress-tests, it’s important to note that they can also be exploited for malicious purposes. However, their primary use remains in benchmarking, integration tests, and thermal tests, where their ability to push the system to its limits helps evaluate its performance and endurance. These tests are essential for identifying potential vulnerabilities, ensuring system stability, and assessing the effectiveness of cooling solutions in managing heat dissipation.

3.4 What is GeST

GeST is an automatic framework designed for generating stress-tests. Developed in Python 3, GeST utilizes XML files for configuring its parameters. The framework consists of five key components: inputs, outputs, a generic algorithm engine, measurement, and the evaluation of measurements.

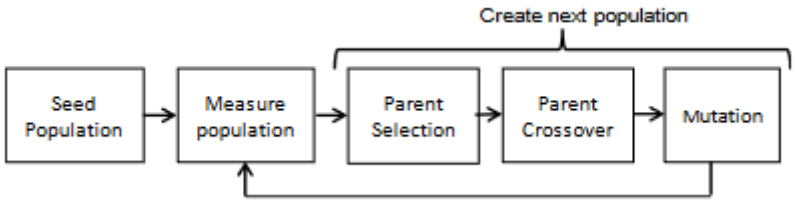

Figure 3.4.1 Steps of GA algorithm

The most crucial component of GeST is the generic algorithm (GA) engine. This engine is responsible for selecting the fittest program based on measurements. It employs genetic algorithms inspired by natural evolution to evolve the programs over successive generations. The process begins by generating an initial seed population of assembly instruction programs. This initial generation can be randomly generated or derived from previous runs. Each individual in the population is then compiled and executed, and a fitness value is assigned based on a chosen metric.

The next step involves creating the next generation based on the Tournament selection. The fittest programs from the previous generation are selected as parents and undergo a crossover process. This process involves recombining the instructions of the parent programs to produce children. Also, the best individual of each generation always progresses to the next generation, this is called elitism. Additionally, a mutation process occurs where random instructions or operands within a child program are altered based on user-defined mutation rate. This mutation rate determines the frequency of mutations during the search. The figure 3.4.1 below illustrates the creation of a new generation using simple viruses as an example.

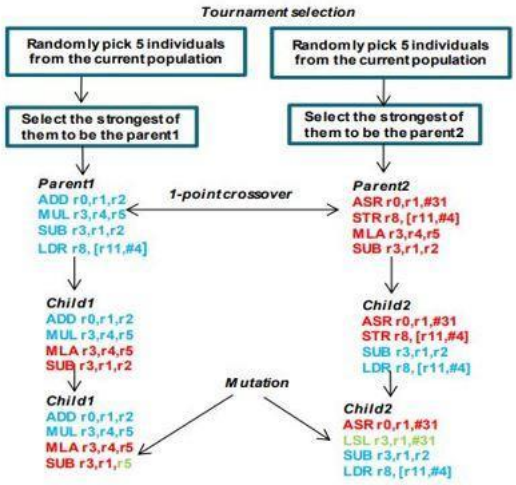

Figure 3.4.1 Crossover and Mutation process

This procedure continues until the desired population size is reached. Throughout the search, inputs are specified through XML files, particularly the configuration file. This file defines various inputs such as population size, number of loop instructions, mutation rate, type of crossover operation, whether elitism is employed, number of parent programs, instructions and operands used in the GA search and the output directory.

Parameter	Default Values
population_size	50
Individual Size (number of loop instructions)	15-50
mutation_rate	0.02 - 0.08
crossover_operator	one point crossover
elitism (Best individual promoted to next generation)	TRUE
parent_selection_method	Tournament Selection
tournament_size	5

Figure 3.4.2 Default GA parameters

Measurement and fitness evaluation are performed using a measurement file written in Python. This file outlines the process of compiling and running the measurement metric, setting the fitness parameter with the metric value, and evaluating it using a function. Individuals with the best fitness values are then selected as parents for the next generation.

Lastly, GeST generates output files including saved text files for all individuals, identifies by generation number, individual ID, and an array of measurements. Additionally, the framework provides a binary file that can be loaded into a Python script to obtain advanced results such as statistical analysis of the fitness value of the fittest individual per generation and an instruction mix breakdown of the fittest individual per generation.

One significant drawback of this procedure is the time-consuming nature of calculating power-consumption by compiling and running each individual.

Generation of program	Compile	Run	Gathering metrics
0.018 seconds	2 seconds	5 seconds	0.018 seconds

Figure 3.4.1 times of processes in normal search

The time required for each step involved in generating a powerful power virus using normal search is illustrated in Figure 3.4.1. These steps outline the time needed for a single individual to complete the process. Taking these time estimates into consideration, to produce 50-70 generations, with each generation consisting of 50 individuals, would require approximately 5-7 hours.

3.5 QuickGen Method

The QuickGen method is a technique developed to produce simple and powerful power viruses quickly. It was created as a solution to the time-consuming process of compiling and gathering power consumption for each individual virus in a search. Instead of compiling and running metrics on each individual, the QuickGen method utilizes the statistical approach of Euclidean distance.

The method involves using the features of a powerful power virus and incorporating them into the Euclidean distance formula. The characteristics of the most powerful virus in the search are used as reference values in the formula.

To ensure consistent comparisons, the values of each characteristic are normalized. Two methods of normalization are applied. The first method normalizes the number of individual types of instructions by dividing the total amount of a specific instruction type (e.g., scalar instructions) by the total number of instructions. The second method normalizes the total number of instructions using the formula: Abs(1 – (Total_under_study / Total_reference)),

where “Abs” denotes the absolute value, Total_under_study represents the total instructions of the individual under study, and Total_reference represents the total instructions of the reference virus.

The following characteristics were found to be suitable for the search using the QuickGen method:

- Number of scalar instructions (normalized)
- Number of load or store instructions (normalized)
- Number of scalar load or store instructions (normalized)
- Number of vmultpd instructions (normalized)
- Number of vaddpd instructions (normalized)
- Number of vmaxpd instructions (normalized)
- Number of vsubpd instructions (normalized)
- Number of vxor instructions (normalized)
- Normalized total amount of instructions

Generation of program	Gathering metrics
0.038 seconds	0.038 seconds

Figure 3.5.1 times of processes using Quickgen Euclidean distance search

Figure 3.5.1 presents the time requirements for each step in the QuickGen method when generating an individual. Remarkably, it takes only 0.038 seconds to generate a single individual. Considering the generation of 50-70 generations with 50 individuals per generation, the total time needed is approximately 1.5 – 2 minutes. Once all the individuals are generated, it is recommended to compile and collect power consumption metrics for the first 10 individuals with the lowest Euclidean distance. Extensive experimentation has shown that this value yields high power consumption. However, this approach does come with a trade-off, as running 10 individuals can take anywhere from half an hour to an hour. Nevertheless, the QuickGen method enables the execution of more searches.

It’s important to note that the QuickGen method relies on having a reference virus to obtain characteristics. This reliance on a reference virus represents a major drawback of the procedure, as it introduces a non-automated aspect to the process. Addressing this limitation is the primary focus of my thesis, aiming to integrate a procedure that automatically identifies the features of the reference virus and incorporates them as input for the QuickGen method.

The reference virus for QuickGen method can be found in the following ways. Firstly, it can be obtained from a publicly available virus known to have high power consumption. Alternatively, in our research, we derive the reference virus by running GeST methodology for multiple generations and selecting an individual (stress-test) with high power consumption.

Chapter 4

GeST-R Method

4.1 Idea

Considering the limitation of the QuickGen method, which relies on having a reference virus, I have proposed a novel approach to address this issue in my thesis. To overcome the need for a predefined reference virus, I have developed a methodology that utilizes the power of regression models and the iterative search process of the GeST framework. The methodology involves running the GeST algorithm for a smaller number of iterations, typically around 10-15 generations, and collecting the data from these generations. This data is then used to train regression models that can accurately predict the optimal reference virus features for future generations. By analyzing the trends and patterns in the collected data, the regression models can provide valuable insights into evolution of the features over time. This approach allows for a more dynamic and adaptive generation of powerful power viruses, as the regression models enable the prediction of optimal features even after substantial number of generations, such as the 50^{th} generation. This innovative methodology not only eliminates the dependency on a predefined reference virus but also enhances the efficiency and effectiveness of the overall search process.

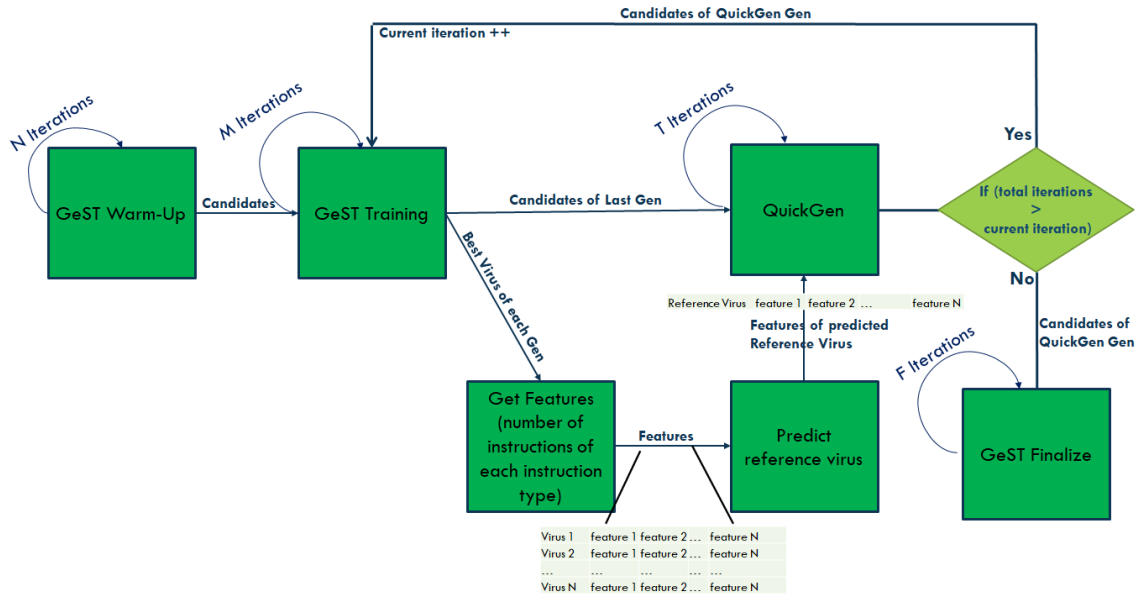

Figure 4.1.1 GeST-R control flow schema

The control flow schema of the GeST-R methodology is as follows:

1. Warm-up for N normal GeST iterations: This initial step involves running a warm-up phase for specified number of normal GeST iterations. During this phase, each individual is compiled and measured for power consumption. The purpose of this warm-up phase is to exclude the initial iterations from the regression training dataset. Through experimentation, it has been observed that the features in the early iterations may not be as reliable as those in later iterations.
2. Training for the next M normal GeST iterations: Following the warm-up phase, the methodology proceeds to run normal GeST for M iterations. Similar to the warm-up phase, each individual is compiled and measured for power consumption. The data from this step is then utilized to train the regression models, which they will serve as a predictive tool for determining the optimal reference virus features in future generations.
3. Collection of data from training phase: During this step, the methodology collects data on the features of each generation. Specifically, it records the number of instructions of each instruction type for the individual with the highest power consumption in each generation. This data serves as the foundation for training the regression models and capturing the trends and patterns in the evolving features.
4. Prediction of reference virus based on collected features: Using the trained regression models, the methodology predicts the optimal reference virus for subsequent iterations. The regression models leverage the patterns and correlations identified in the training phase to make accurate predictions regarding the most suitable features for maximizing power consumption. It is worth mentioning that the framework tests multiple

logarithmic regression models (ln(x), log2(x), log10(x)), and chooses the one with the lowest error with cross-validation methodology. Specifically, the training set is divided in 5 parts (folds). During the error testing of the regression models, 4 parts out of 5 are used as training sets, and the remaining one is used as validation set. This process is being repeated for 5 times and each time a different validation set is chosen. At the end, the error of the regression model is the average error calculated from the cross-validation procedure. Finally, the chosen regression model will be the one with the lowest error.

5. Execution of QuickGen method for T iterations: With the predicted reference virus in hand, the methodology proceeds to run the QuickGen method for a specified number of iterations, denoted as T. Rather than compiling and running each individual to gather power consumption, the methodology utilizes the concept of Euclidean distance from the reference virus. The top 10 individuals with the lowest Euclidean distance are compiled and run, as it is important to note that low Euclidean distance alone does not guarantee high power consumption. Since the features used to calculate the Euclidean Distance may not fully capture all the characteristics that define a virus with high power consumption, it is important to acknowledge that this metric alone may not be sufficient. However, the observed trends indicate a correlation between the Euclidean Distance with the instruction types and the power consumption.
6. In the final step, there are two options. The first option is to proceed with running normal GeST iterations for a specified number of F iterations and conclude the procedure. This is essential because Euclidean distance alone is not a definitive indicator of high-power consumption. The second option is to restart the entire process, beginning again from step 2. This choice enables iterative repetition of the methodology, allowing for continuous improvement and refinement of the prediction and generation process. By reapplying the training phase and leveraging the regression models, it becomes possible to adapt and enhance the accuracy of prediction the reference virus features for future generations.

4.2 The Power of Regression Models in Virus Generation

In this section, we present a compelling rationale for utilizing regression models in our virus generation methodology. Through a series of experiments, we conducted a comprehensive analysis of the relationship between instruction types, power consumption, and the effectiveness of regression models. Our approach involved running normal GeST for 50 generations, collecting data on the number of instructions of each instruction type. We then utilized the data from the first 20 generations, excluding the initial 10 as warm-up iterations, and used the subsequent 10 generations for training our regression models.

Upon applying the trained regression models to predict the outcome of generation 50, we compared the predicted number of instructions of each instruction type with the actual outcomes obtained from normal GeST. Remarkably, we found a striking similarity between the regression model predictions generated by the regression models and the observed outcomes of normal GeST. This will help us build the good power virus features for QuickGen method.

After conducting the experiments and training the regression models, we compared the predicted number of instructions of each instruction type from the regression models with the actual results from the normal GeST generations. To visually illustrate these findings, we have included Figure 4.2.1 which shows the comparison between the predicted and the actual number of instructions for each type.

The graphs clearly demonstrate the close alignment between the predictions generated by the regression models and the actual results obtained from the normal GeST iterations. These results validate the effectiveness and reliability of using regression models for predicting the good reference virus features based on instruction types. In other words, regression models, manage to predict the features of a virus that needed 50 generations of normal GeST procedure at just 20 generations. Regression models are able to replace some of the GeST generations but not all of them, because we need some generations to train them. Also, the regression models predict some of the features and not all of them, therefore GeST is needed for exploring the uncontrolled features (e.g., dependencies).

Figure 4.2.1 Accuracy of the Regression Models

4.2 Integration of GeST-R

In this section, we will discuss the successful integration of GeST-R, our developed methodology based on regression models, into the GeST framework. By incorporating regression models, we aimed to introduce reference virus prediction for QuickGen method. This integration enables a fully automatic and faster procedure for the effective generation of power viruses.

In terms of input/output, GeST-R takes input from the GeST configuration file, where parameters such as the number of warm-up, training, QuickGen, and finalize iterations are specified. Additionally, the configuration file includes the Total GeST-R iterations and the GeST-R flag, which determines whether GeST-R or normal GeST should be executed.

The functionality of GeST-R is exposed through an API that allows seamless interacting with the GeST framework. This API provides access to key functions for training the regression models, predicting the reference virus, and evaluating the fitness of individuals during different stages of the evolutionary process.

One important function is the Reference_Features function, which takes the instruction type data from the training generations. This data is used to train the regression models, enabling them to predict the reference virus based on the number of instructions of each instruction type. The prediction function then utilizes the trained models to predict the reference virus for a future generation.

During the warm-up and training iterations, the fitness evaluation is based on power consumption, which is measured by compiling and running each individual virus. For QuickGen iterations, the fitness evaluation involves calculating the Euclidean distance between the current individual and the predicted reference virus using the trained regression models. Finally, during the finalize iterations, the fitness is once again determined by measuring the power consumption of each individual virus.

By integrating GeST-R into GeST, we have introduced a more sophisticated and efficient approach to power virus generation, leveraging the power regression models for accurate prediction.

Chapter 5

Experimental Methodology

5.1 Hardware Configuration

This research is conducted on a single machine, namely the HP Compaq 8200 Elite CMP PC, which is equipped with an Intel® Core™ i5-2400 CPU @ 3.10GHz processor. The CPU consists of 4 cores with a base frequency of 3.10 GHz and a thermal design power (TDP) of 85 Watts. To ensure stability, the CPU frequency was regulated using the cpupower tool, which allows the operating system to dynamically adjust the CPU frequency based on power-saving or performance-enhancing needs. This tool enables automatic scaling in response to system load, as well as manual adjustments by user space programs ("CPU frequency scaling – ArchWiki"). The processor is equipped with 6MB of cache and operates at a bus speed of 5 GT/s. When examining the microarchitecture of the processor, it is observed that the L1 instruction cache can handle 16 bytes per cycle, and there is a 4-way decode capability. Furthermore, this microarchitecture comprises 6 execution units. To ensure consistent results, c-stats and p-states were disabled during all experiments to eliminate variations. It is important to note that the generated viruses solely stress the CPU, as other components of the system such as DRAM frequency were not configured.

5.2 GeST and Viruses

The GeST tool served as the primary workload for generating and evaluating power viruses in this research. The GeST-R methodology, which integrates regression models into the GeST framework, was employed to enhance the generation of power viruses. GeST-R introduced a reference virus prediction approach, leveraging regression models to predict the number of instructions for each instruction type.

The GeST configuration file was updated to include additional parameters specific to GeST-R. These parameters included N for warm-up training iterations, M for training iterations, T for QuickGen iterations, F for finalize iterations, the total number of GeST-R iterations, and the GeST-R flag. By setting the GeST-R flag to 1, GeST-R was executed, while setting it to 0 resulted in the normal GeST procedure.

During the fitness evaluation process in the GeST-R methodology, several steps were undertaken. Initially, power consumption was utilized as the fitness value for the warm-up and training iterations. Each individual virus underwent compilation and execution to measure its power consumption, which served as the fitness value. Subsequently, during the QuickGen iterations, the Euclidean Distance between the current individual and the reference virus was calculated and considered the fitness value. The reference virus was determined using the trained regression models based on the training iterations. Finally, following the completion of QuickGen iterations, the power consumption of each individual virus was once again measured during the finalize iterations, and this power consumption value was employed as the fitness value.

Power consumption measurements were obtained by executing each virus for a fixed duration of 5 seconds and recording the corresponding power consumption of the system. These measurements served as a key metric to determine the effectiveness of the generated power viruses.

It is important to note that the version of GeST used in this research is not publicly available. The GeST implementation and its integration with the GeST-R methodology were developed specifically for this study and tailored to the research objectives.

5.3 Power Measurement and Analysis

This research focuses on several CPU-related metrics that are crucial for understanding the impact of programs on the processor. In particular, power consumption serves as a fundamental metric for assessing program performance in the context of power viruses. To measure power consumption, the program was compiled, and the likwid-powermeter tool was employed. Likwid-powermeter (based on rapl counters) is specifically designed for Intel CPUs and provides accurate power and clocking information. By utilizing this tool, the power consumption of the program could be effectively measured in various research scenarios. The analysis of power consumption data played a vital role in evaluating program performance and determining the effectiveness of the developed power viruses.

5.4 Validation Effort

Validation played a crucial role in ensuring the accuracy and reliability of various components in my research, including the regression models, QuickGen method and power consumption of the normal GeST procedure, and the overall GeST-R methodology. Validation measures were implemented to assess the performance and effectiveness of each element.

To validate the regression models, extensive experiments were conducted to compare the predicted outcomes with the actual results obtained from the normal GeST procedure. This involved analyzing the number of instructions for each instruction type and evaluating how closely the regression models aligned with the observed trends.

For the QuickGen Method, validation was carried out by running multiple iterations and analyzing the generated power viruses. The power consumption metrics were meticulously measured and compared to ensure that QuickGen Method produced accurate and consistent results.

The entire GeST-R methodology underwent comprehensive validation to verify its effectiveness in achieving the desired objectives. This involved examining the integration of regression models, the prediction accuracy of reference viruses, and the overall efficiency of the approach in generation power viruses.

By employing validation procedures for each component, I could confidently establish the accuracy, reliability, and effectiveness of the regression models, QuickGen Method, and the GeST-R methodology as a whole.

Chapter 6

Experimental Evaluation

6.1 QuickGen Validation

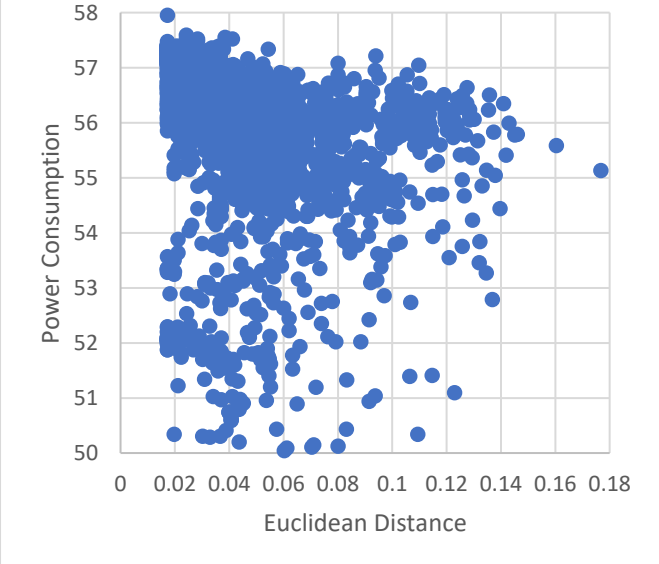

Figure 6.1.1 Euclidean vs Power Consumption

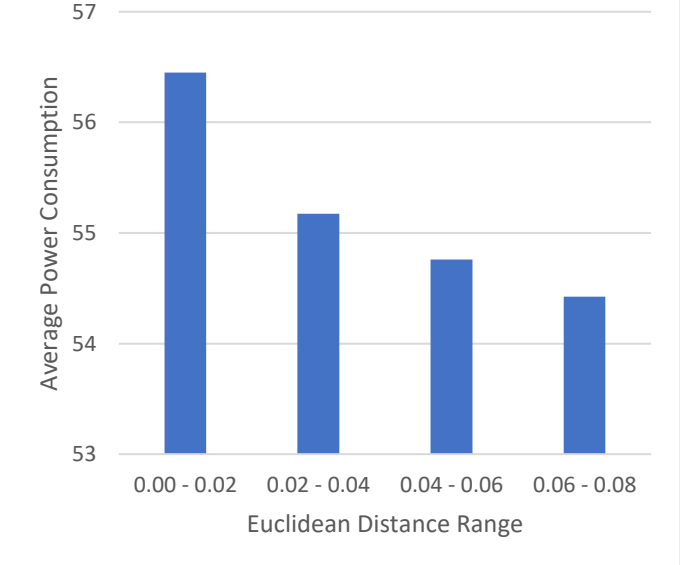

Figure 6.1.2 Euclidean vs Avg Power Consumption

The Figures 6.1.1 and 6.1.2 illustrate the relationship between Euclidean distance and Power Consumption. It is evident that there is a clear correlation between the two variables. As the Euclidean distance decreases, indicating a higher similarity between the current individual and the reference virus, the Power Consumption tends to increase. These finding addresses one of

the research questions posed in the thesis, demonstrating the potential of QuickGen in optimizing the execution time of stress-test generation.

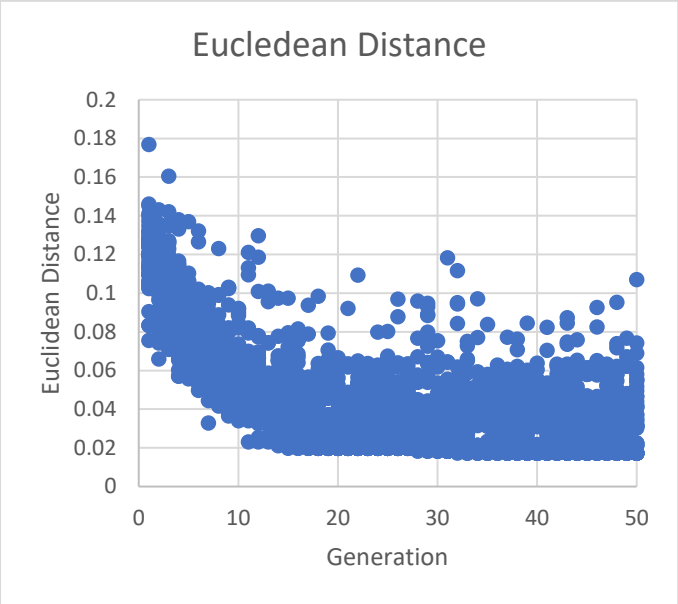

Figure 6.1.3 Euclidean vs Generations

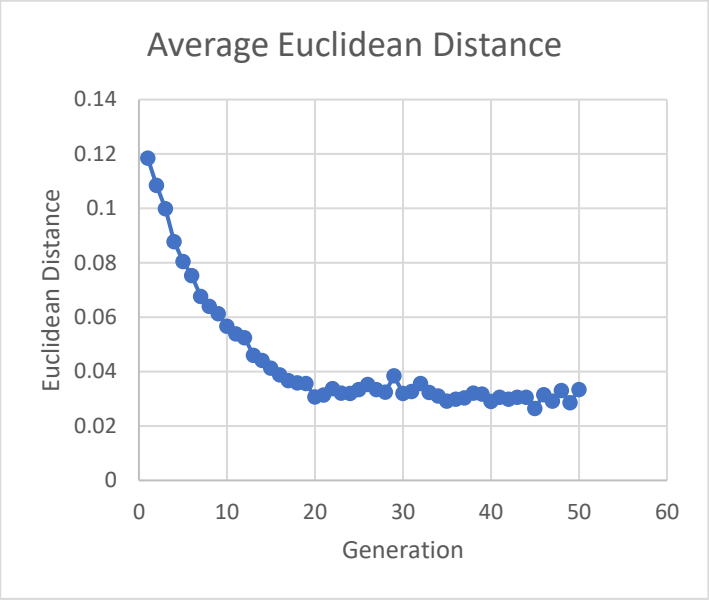

Figure 6.1.4 Avg Euclidean vs Generations

The Figures 6.1.3 and 6.1.4 show the relationship between generation and Euclidean distance. As the generation number increases, indicating the progression of the stress-test generation process, the Euclidean distance between the current individual and the reference virus decreases. The decreasing Euclidean distance signifies an improved similarity between the generated individuals and the desired characteristics of the reference virus. This trend validates the efficacy of the stress-test generation methodology.

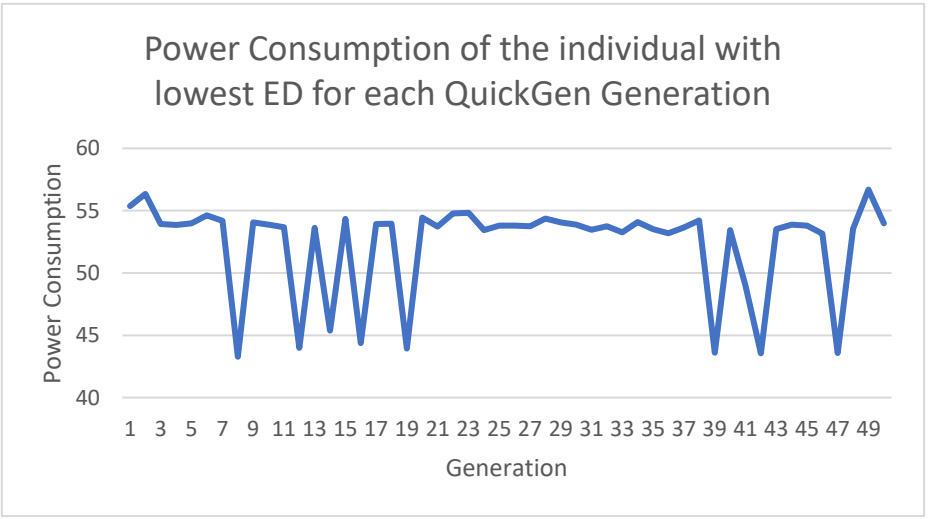

Figure 6.1.5 Power Consumption vs QuickGen generations

During the analysis of the figure 6.1.5, an important finding appeared in the GeST-R methodology regarding the relationship between Euclidean Distance and power consumption. The figure shows that lower Euclidean Distance does not always correspond to better power consumption. This can be attributed to the methodology's focus on a limited set of features, the number of each instruction type, while overlooking other crucial factors like dependencies. As a result, the correlation between Euclidean distance and power consumption exhibited fluctuations. The fluctuations are also evident from the first graph (Euclidean distance vs Power Consumption), at 0.02 Euclidean distance there are viruses with 50W power consumption and others with 58W. To overcome this issue, a solution was to measure the power consumption of the top 10 individuals with the lowest Euclidean distance at each QuickGen generation. Then the tournament selection will be among those 10 individuals, and it will be based on their power consumption. This additional step successfully addressed the fluctuating power consumption problem, and promoting individuals with high-power consumption to the subsequent generations.

After incorporating the additional step of measuring power consumption for the top 10 individuals with the lowest Euclidean distance at each QuickGen generation, the results demonstrate notable improvements in the power consumption of generated viruses.

6.2 GeST-R Parameter Analysis

6.2.1 QuickGen generations vs Power Consumption

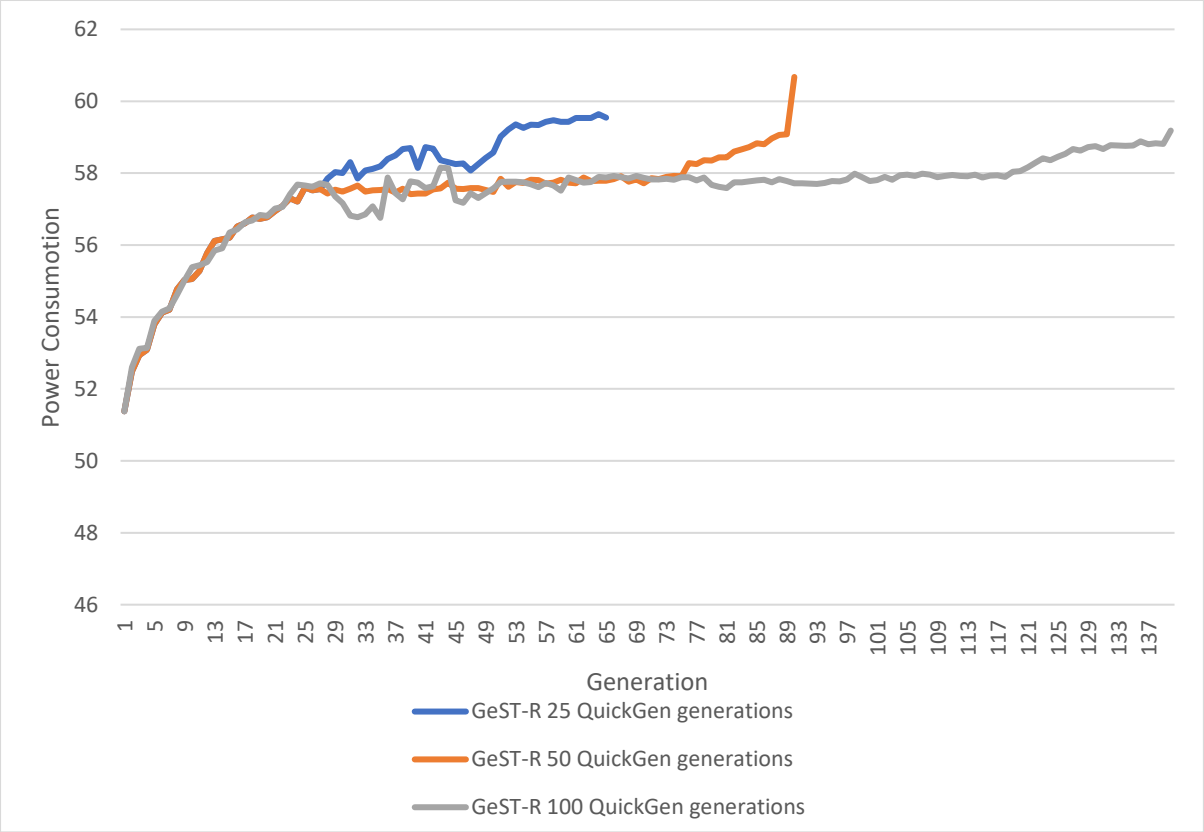

Figure 6.2.1.1 QuickGen Generations vs Power Consumption

Based on the experimental results showed in the figure 6.2.1.1, where GeST-R was evaluated with 25, 50 and 100 generations of QuickGen, it was observed that the number of QuickGen generations has a significant impact on power consumption and computational time. The graph demonstrated that increasing the number of QuickGen generations beyond 25 did not result in substantial improvements in power consumption. In fact, viruses generated with 25 generations of QuickGen exhibited comparable power consumption and sufficiently low Euclidean distance compared to those generated with a higher number of generations. Additionally, using 25 generations of QuickGen led to a significant reduction in computational time compared to using 50 generations. This reduction can be attributed to the time-consuming nature of measuring the power consumption of 10 individuals for each QuickGen generation. Considering the trade-off between power consumption and computational time, the results suggest that 25 generations of QuickGen, coupled with 15 training generations, 10 warm-up generations, and 15 finalize generations, provide an optimal balance of efficiency for the GeST-R methodology.

The possible reasons why there is no improvement in power consumption after 25 generations, could be attributed to the following factors. Firstly, it is likely that the features characterizing

the predicted virus are already reached within the initial 25 generations, making further iterations unnecessary. Secondly, Euclidean distance alone may not capture all the relevant features that characterize the best virus, requiring additional iterations with power consumption measurement to uncover the remaining features. Lastly, it is possible that the optimization process gets trapped in a local optimum, preventing further improvements beyond 25 generations.

During the experimentation, it was observed that reducing the number of QuickGen generations below 25 resulted in a notable decrease in the quality of generated individuals. Specifically, viruses generated with fewer than 25 generations of QuickGen did not have low Euclidean distance compared to the reference virus. This indicated that a higher number of QuickGen generations is necessary to ensure the selection and evolution of individuals with closer feature characteristics to the predicted reference virus. Thus, the finding highlight the importance of maintaining a minimum threshold of 25 generations to achieve desirable results in terms of Euclidean distance and feature similarity.

6.2.2 Training generations vs Power Consumption

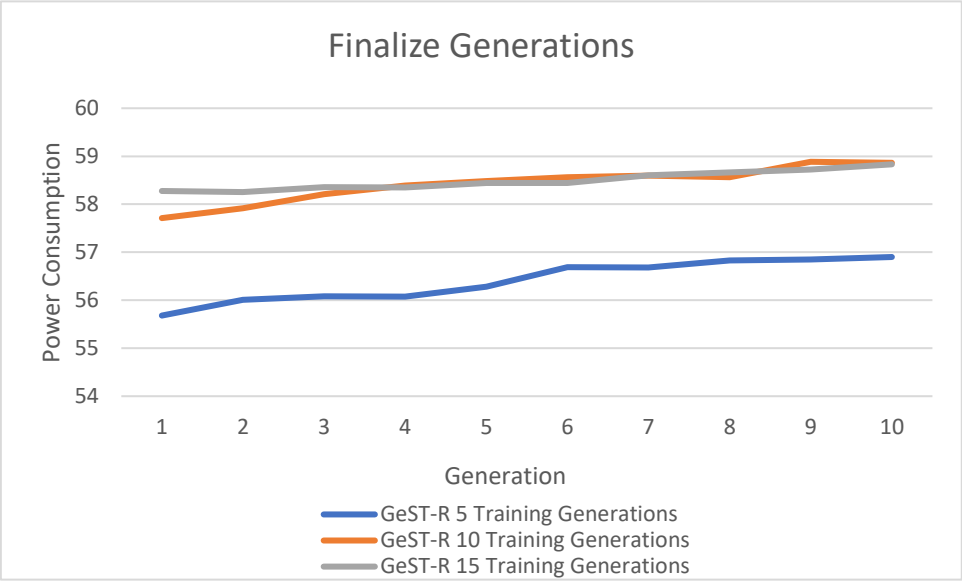

Figure 6.2.2.1 Power Consumption vs Finalize Generations

Based on the experimental results, the figure 6.2.2.1 shows that the number of training generations in the GeST-R methodology has a significant impact on the accuracy of the predicted reference features. The experiments compared the performance of GeST-R with 5, 10 and 15 training generations, focusing on the last 10 generations of finalize iterations, the last step of GeST-R procedure. It was found that 5 training generations were insufficient to effectively train the regression model, resulting in lower-quality predictions of the reference virus features. However, with 10 and 15 training generations, the regression models were well-trained and produced accurate predictions of the reference virus features. Interestingly, the difference in performance between 10 and 15 training generations was minimal, suggesting that the additional 5 training generations did not provide significant improvements in the quality of the predictions. Considering the additional time required for 5 more training generations, it was concluded that 10 training generations offer a more balanced trade-off accuracy and computational efficiency in the GeST-R methodology.

6.2.3 Warm-up generations vs Power Consumption

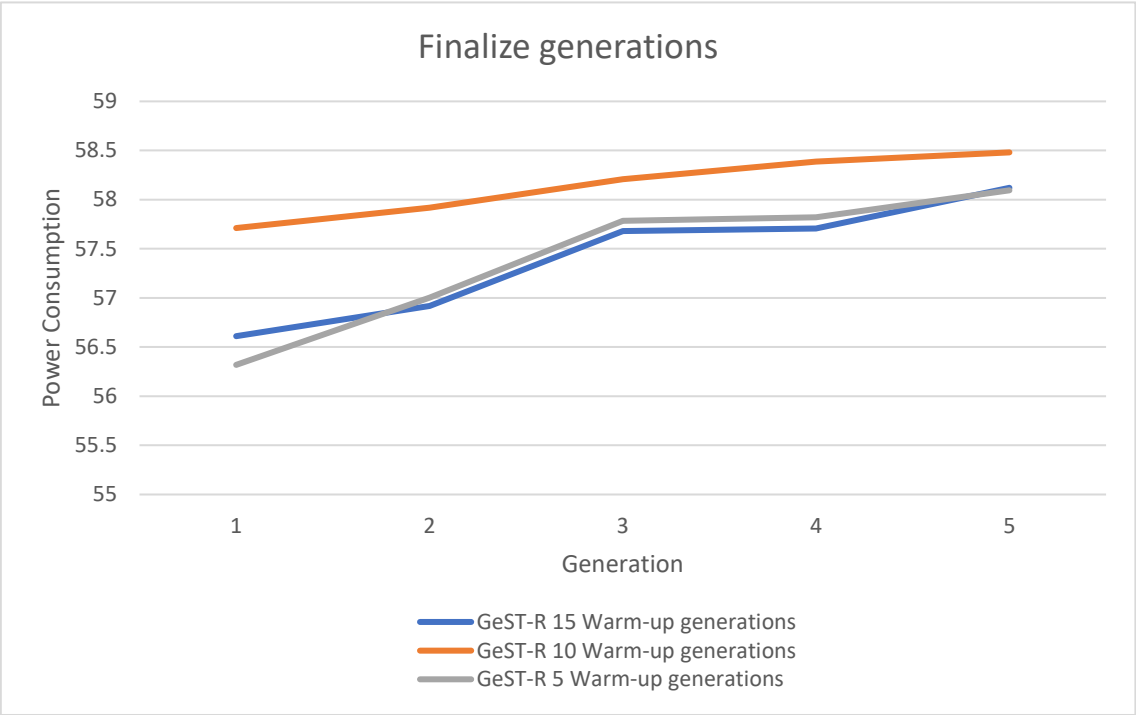

Figure 6.2.2.1 Power Consumption vs Finalize Generations

The experimental evaluation of GeST-R with different values of warm-up iterations (5, 10, and 15) revealed interesting insights into the power consumption trends during the Finalize step. The figure 6.2.2.1 shows a lower power consumption for the individuals in the first generations of the Finalize step for both 5 and 15 warm-up iterations compared to the warm-up 10. The observed trend can be explained by the importance of having sufficient number of warm-up iterations to gather valid data for training the regression models. With fewer warm-up iterations, the early generations of GeST may not provide accurate data, leading to non-optimal regression models. On the other hand, excessive number of warm-up iterations can potentially lead to the exclusion of important data, such as significant changes between early generations, thereby impacting the quality of the regression model. Based on the results, it was determined that 10 warm-up iterations strike the optimal balance, allowing for the generation of reliable regression models and ultimately improving the power consumption of the final individuals.

6.2.4 Finalize generations vs Power Consumption

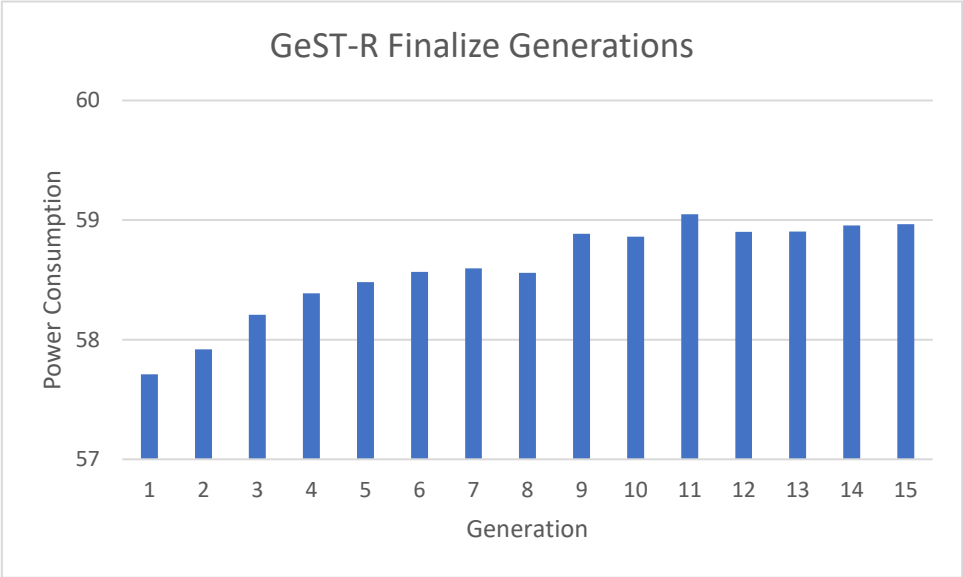

Figure 6.2.3.1 Power Consumption vs Finalize Generations

By analyzing the figure 6.2.3.1 showing the relationship between finalize generations and power consumption, a clear trend emerges. The graph, which includes 15 finalize generations, demonstrates that the power consumption stabilizes after around 10 generations. Beyond this point, the difference in power consumption between successive generations becomes increasingly small. These results indicate that 10 generations are sufficient to achieve a good outcome in terms of power consumption. Additional finalize generations beyond this point will add more time delay. Therefore, it can be concluded that 10 finalize generations provide a practical and efficient choice for achieving desirable power consumption results in the GeST-R methodology.

6.2.5 Repetitive Runs vs Power Consumption

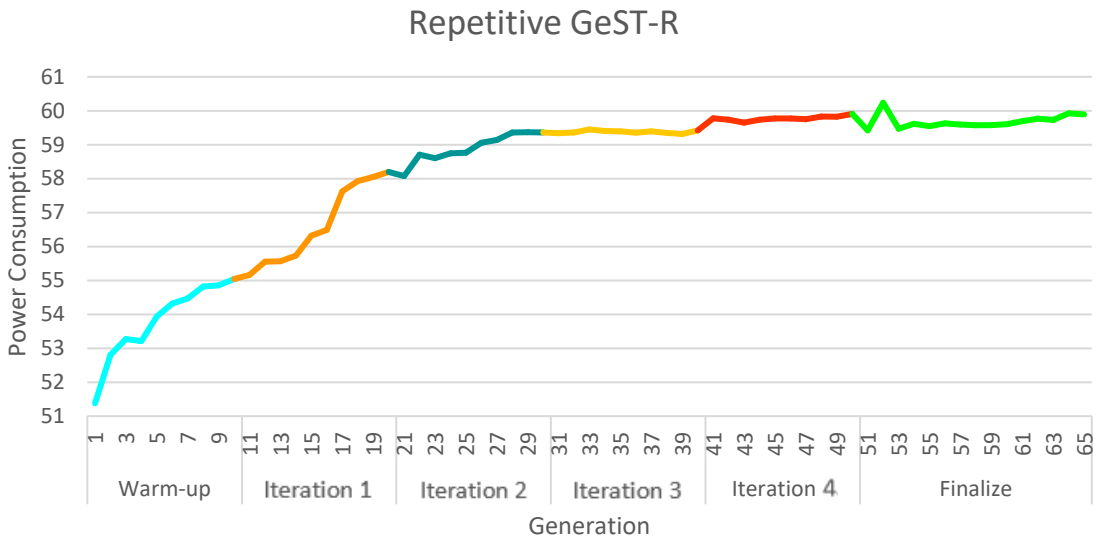

Figure 6.2.4.1 Iterative GeST-R procedure

As a solution to the issue of getting trapped in a local minimum during the optimization process, a repetitive GeST-R procedure was introduced. The main idea behind this procedure is to train multiple times during the whole procedure the regression models, allowing for the discovery of new trends that may have been missed by the initial data which we used to train regression models. This approach aims to overcome the limitations of local minimum by exploring different directions in the optimization process.

The figure 6.2.4.1 presented demonstrates the results of the GeST-R repetitive procedure. To execute the repetitive procedure, the gestr_iterations parameter in the configuration file needs to be adjusted to the desired number of iterations. Each iteration comprises a training phase, regression model training based on the training phase, and the QuickGen method. After all iterations are completed, the finalize step is executed.

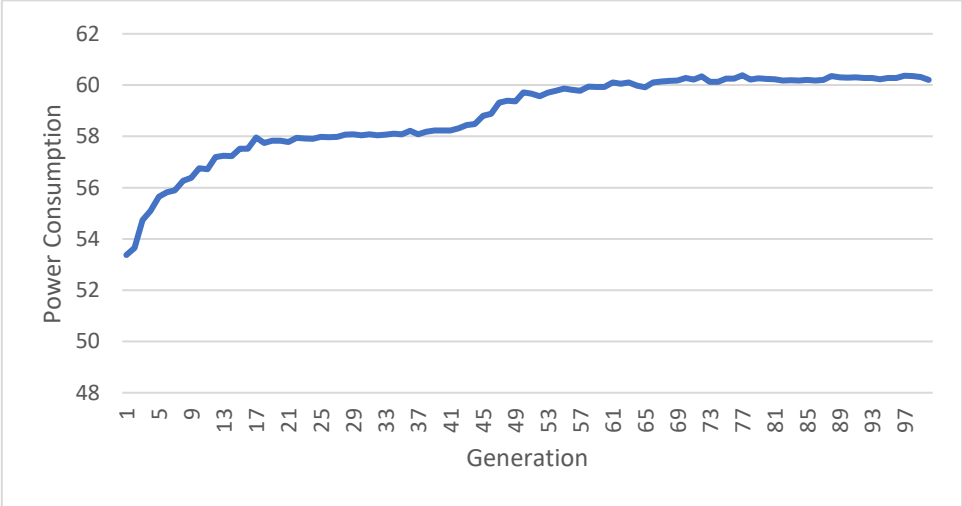

Figure 6.2.4.2 Power Consumption vs Generations

The figure 6.2.4.1 reveals that significant improvements can be observed after the first iteration of the repetitive procedure. However, subsequent iterations show only slight improvements. This can be attributed to reaching the limit of power consumption for the specific processor being used. To determine the power consumption limit, the normal GeST procedure was run for 100 generations, revealing that it reaches a limit approximately 60W for two instances as the following figure 6.2.4.2 shows.

It is important to note that the figure 6.2.4.1 does not include the power consumption of the QuickGen generations. This is because of the significantly reduced time required to execute a QuickGen generation compared to a regular GeST generation.

6.3 GeST-R vs GeST

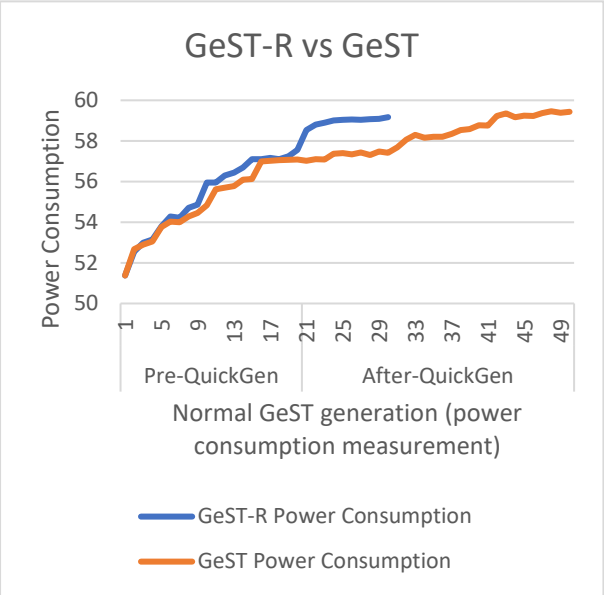

Figure 6.3.1 Power Consumption vs GeST generations

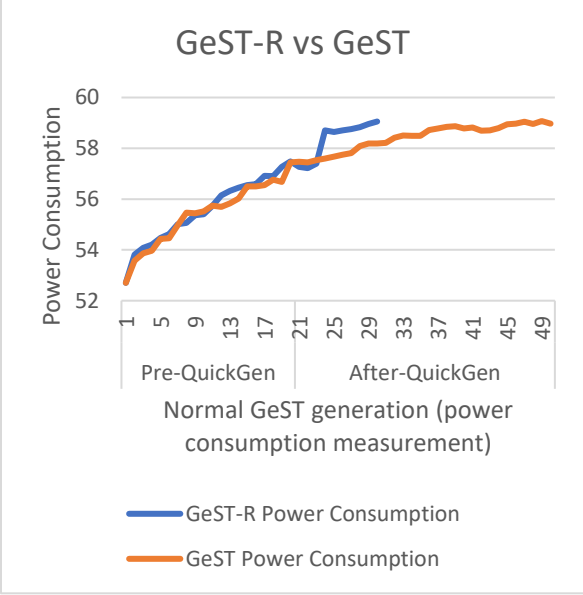

Figure 6.3.2 Power Consumption vs GeST generations

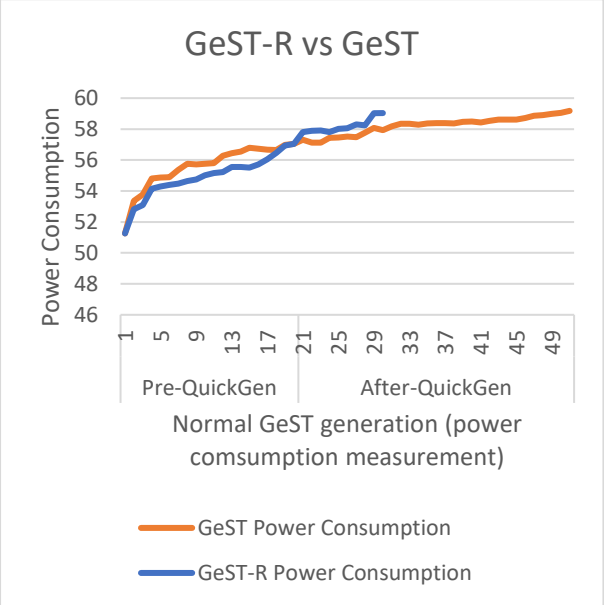

Figure 6.3.3 Power Consumption vs GeST generations

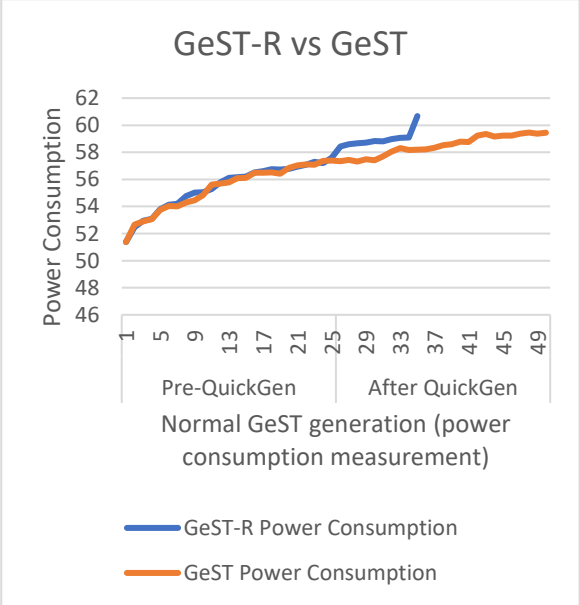

Figure 6.3.4 Power Consumption vs GeST generations

The figures 6.3.1, 6.3.2, 6.3.3 and 6.3.4 are showing the comparison between GeST-R and GeST methodologies in terms of power consumption. The y-axis represents the power consumption of the individuals, while the x-axis represents the number of generations. The power consumption of the highest-power individual from each generation is plotted for both GeST-R and GeST. The starting generation of the graphs is the same between the GeST-R and GeST methodologies, however the figures 6.3.1, 6.3.2, 6.3.3 and 6.3.4 have different starting generations to validate the GeST-R procedure (that is working for more cases).

In this comparison, the graphs are clearly showing that the final produced individual of the GeST-R produces power consumption levels that are nearly equal to those of the GeST 50^{th} generation individual. This demonstrates the effectiveness of the GeST-R methodology in optimizing the stress-test generation process.

It is important to note that the graphs do not include the power consumption of the QuickGen generations. This is because of the significantly reduced time required to execute a QuickGen generation compared to a regular GeST generation. As such, the 25 generations of QuickGen are equivalent to just 5 generations of GeST in terms of time.

The figure 6.3.3 reveals interesting differences between the two procedures. While they start from the same initial generation, the subsequent generations diverge. This can be attributed to the different used operators (cross-over, mutation) and they produced different individuals. GeST-R initially has lower power consumption, but after QuickGen method, it surpasses GeST and ultimately produces an individual with power consumption similar to GeST 50^{th} generation.

The figure 6.3.4 has 15 training iterations instead of 10. Increasing the training generations in GeST-R did not significantly change the results, as we showed before. GeST-R still produced an individual with power consumption comparable to GeST's 50^{th} generation. Interestingly, at the final generation of the Finalize step, GeST-R even outperformed GeST by generating an individual with higher power consumption, however this appears to be random rather than attributed to any specific facor.

Overall, the graphs are showing that GeST-R effectively use the power of regression models to improve the power consumption of stress-test individuals, offering a more efficient approach to stress-test generation.

6.4 GeST individual optimization

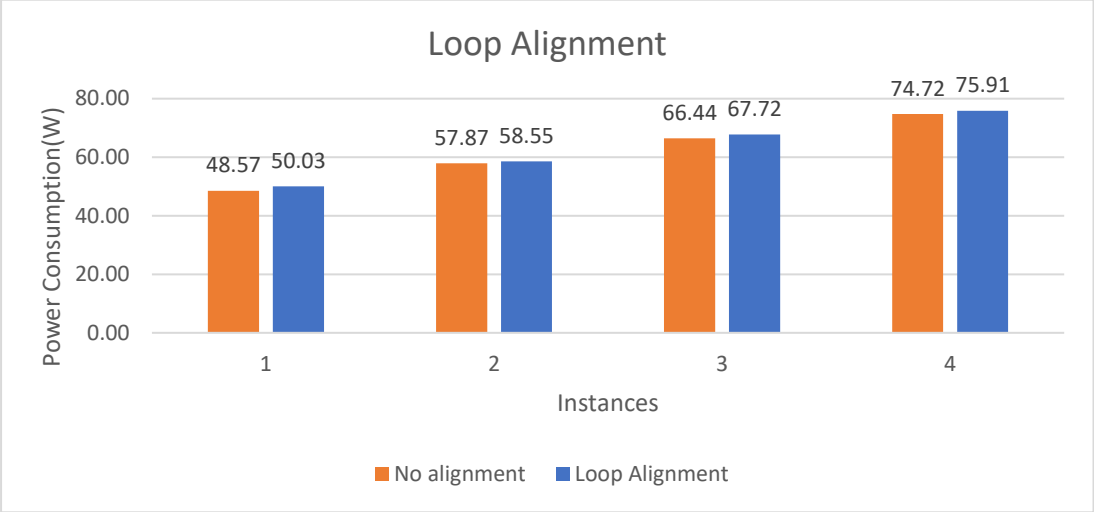

Figure 6.4.1 Power of Loop Alignment vs No Loop Alignment

To evaluate the impact of loop alignment on power consumption, I conducted an experiment comparing the power consumption of the current GeST individual with an individual with an aligned loop address (the individuals were the same). Loop alignment involves placing the loop start address on a new cache block, potentially reducing cache misses and improving power consumption. The figure 6.4.1 in the evaluation illustrates the power consumption results for both cases, revealing that the individual with loop alignment resulted in a slightly higher power consumption for all instances compared to the non-aligned approach.

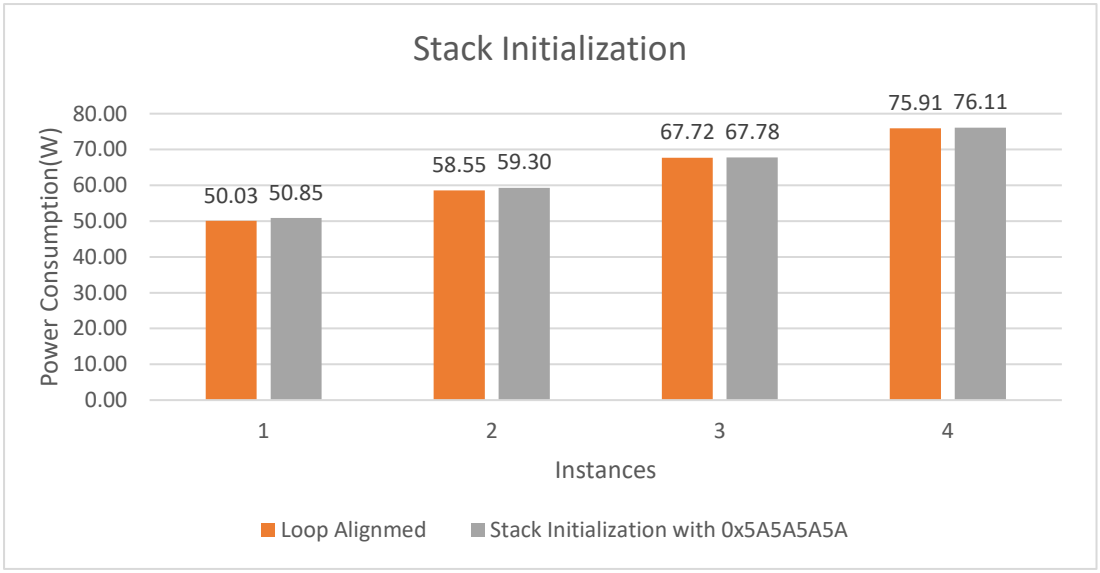

Figure 6.4.2 Power of Stack Initialized vs No Stack Initialized

The figure 6.4.2 presents the experiment conducted on stack initialization, where the stack was initialized with values to increase CPU stress during instruction execution. Interestingly, the results indicate that the individuals with stack initialization exhibited higher power consumption to those without stack initialization.

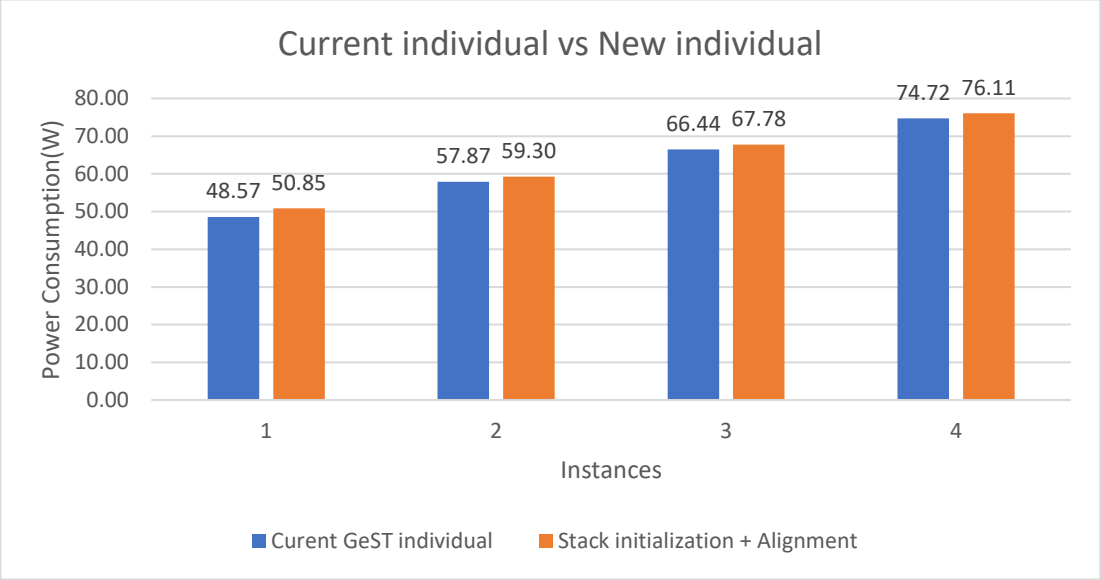

Figure 6.4.3 Power of current GeST individual vs Stack initialized and Loop Aligned individual

Combining the loop alignment and stack initialization improvements, the graph 6.4.3 demonstrates the power consumption improvements achieved by applying these techniques to the same individual. By adding loop alignment and stack initialization to the individuals, we can observe an improvement in power consumption compared to the initial individual (before the changes).

Chapter 7

Related Work

In the field of stress-test generation and optimization, several projects and frameworks have been developed to address similar challenges. One notable project is MicroGrad, which is another automated framework for stress-test generation. MicroGrad utilizes the Microprobe code generation framework and employs a Gradient Descent-based tuning mechanism to evolve test cases for Stress testing. By iteratively adjusting parameters, the tuning mechanism aims to improve performance metrics such as execution time, power consumption, or efficiency. The evaluation of MicroGrad demonstrates its high accuracy and low resource requirements.

In comparison to MicroGrad, GeST-R offers a distinct approach to stress-test generation and optimization. While MicroGrad focuses on utilizing a Gradient Descent-based tuning mechanism to evolve test cases, GeST-R incorporates Genetic Algorithms and regression models for optimizing stress-test generation. GeST-R integrates the use of Euclidean Distance and regression models to predict the characteristics of the “best-virus” and optimize power consumption. Additionally, it should be noted that comparing the execution time between GeST-R and MicroGrad is not straightforward. While GeST-R runs on real hardware, the evaluation of MicroGrad is conducted on a simulator (GEM5). Furthermore, the comparison is based on generation rather than real-time measurements.

Chapter 8

Future Work

In this section, we will explore potential avenues for future research and improvements based on the findings and outcomes of this study. These directions aim to further enhance the effectiveness and efficiency of the methodology proposed in this thesis.

- Evaluating the Necessity of QuickGen: One important question to address is whether the utilization of the QuickGen method is essential or if the regression models alone can provide sufficient accuracy. It would be valuable to investigate the possibility of generating individuals directly with features equivalent to the output of the regression models. This approach could potentially eliminate the need for running the QuickGen method, leading to a more streamlined and efficient process.
- Expanding Virus Features: To enhance the accuracy of the regression model predictions, it is worth considering the inclusion of additional features beyond just the number of instructions. For example, incorporating dependencies and analyzing the distances between them could provide valuable insights into power consumption patterns. Expanding the set of virus features can potentially yield more robust regression models and improve the overall accuracy of reference virus prediction.
- Automated Procedure Optimization: In order to facilitate the application of the methodology in practical scenarios, developing an automated method for determining optimal warm-up, training, QuickGen, and finalization iterations would be highly beneficial. By automating these steps, researchers and practitioners can save significant time and effort in configuring and fine-tunning the procedure, ultimately improving the efficiency and reproducibility of the experiments.
- Error Prediction for Reference Virus Generation: A key aspect to explore is the development of a method that can predict the number of generations for which the regression models will accurately predict the reference virus. This prediction can be based on analyzing the error rates and patterns observed in the regression models' predictions for future generations. By estimating the limitations of the models in advanced, researchers can optimize the selection and usage of regression models, ensuring their effectiveness in predicting reference viruses for a given number of generations.

These future research directions have the potential to further advance the proposed methodology, enhancing its accuracy, efficiency, and automation. By addressing these aspects, researchers can continue to contribute to the field of power virus generation, opening up new possibilities for advanced and impactful methodologies in the future.

Chapter 9

Conclusion

In this thesis, we set out to address several research questions related to optimizing the execution time of stress-test generation using Genetic Algorithms (GAs) with the inclusion of Euclidean Distance (QuickGen) and Regression Models. Additionally, we explored methods to enhance the power consumption of GeST individuals, specifically targeting Sandy Bridge processors.

Regarding the first research question, we have successfully demonstrated the effectiveness of regression models in predicting the characteristics of the "best-virus" after a certain number of generations. Through training the regression models using instruction type data from the initial generations, accurate predictions were made, contributing to the optimization of the stress-test generation process.

Furthermore, our investigation into the validation of Euclidean Distance as a replacement for power consumption as the fitness value of the Genetic Algorithm yielded positive results. The Euclidean Distance metric proved to be an effective measure of similarity between individuals and the reference virus, facilitating accurate fitness evaluation during the evolutionary process.

Moving on to the second research question, we explored various methods to enhance the power consumption of GeST individuals on Sandy Bridge processors. Our findings revealed that aligning the loop address of GeST individuals improved data storage in cache blocks, resulting in higher power consumption.

Overall, this research has contributed to advancing the field of stress-test generation and power virus development. The integration of Euclidean Distance and Regression Models within the GeST framework has proven to be a successful approach for optimizing execution time and enhancing power consumption.

These findings open up new possibilities for further research and development in the field of stress-test generations and power virus optimization. Future work could focus on refining the regression models, exploring additional metrics for fitness evaluation, and investigating the impact of other processor architectures. By continuing to explore those paths, we can further improve the efficiency and effectiveness of stress-test generation techniques and contribute to advancements in computer system testing and evaluation.

Bibliography

- GeST an Automatic Framework for Generating CPU Stress-Tests https://www.cs.ucy.ac.cy/carch/xi/papers/ISPASS_2019_for_webpage.pdf
- Speeding up the process of generating stress tests, Thesis by ANDREAS FRANGOS
- Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture (April 2022, Order Number: 253665-077US) (Chapter 3 and 4 link: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html)
- FlexWatts: A Power- and Workload-Aware Hybrid Power Delivery Network for Energy-Efficient Microprocessors MICRO2020 (only background section, explain the different power delivery network architectures)
- COMPUTER ARCHITECTURE TECHNIQUES FOR POWER-EFFICIENCY, Stefanos Kaxiras (only chapter 1, explain the dynamic and leakage power and the parameters affecting them, thermal runaway, thermal throttling)
- MicroGrad: A Centralized Framework for Workload Cloning and Stress Testing ISPASS, 2021