LARGE-SCALE OPTIMIZATION OF ELECTRIC VEHICLES USING GRAPHICS PROCESSING UNITS

A DISSERTATION SUBMITTED TO THE GRADUATE DIVISION OF THE UNIVERSITY OF HAWAI‘I AT MĀNOA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

MECHANICAL ENGINEERING

MAY 2012

By
Volker Schwarzer

Dissertation Committee:

Reza Ghorbani, Chairperson
Richard Rocheleau
Weilin Qu
James Griffin
Zachary Trimble

To my family.

Acknowledgments

This research project would not have been possible without the support of many people. First and foremost, I would like to express my thanks and appreciation to my dissertation advisor, Dr. Reza Ghorbani, who was abundantly helpful and offered invaluable assistance, support and guidance.

My deepest gratitude goes to my second advisor and committee member, Dr. Richard Rocheleau, for giving me the opportunity and support to pursue a doctoral degree at the University of Hawai‘i at Mānoa. His broad knowledge of the field and sharp analyses were indispensable contributions to my research.

I would also like to express my sincere appreciation to the members of my supervisory committee, Dr. Weilin Qu, Dr. Zachary Trimble and Dr. James Griffin for their time, effort and valuable suggestions.

Special thanks goes to Dr. David Garmire for providing access to HOSC's GPU cluster. I am also grateful to my fellow student Stephan Fabel for supporting me with his great programming knowledge and computer skills.

Finally, I would like to thank my labmate and friend, Tyler Thornbrue, for proofraeding this dissertation.

Abstract

Large-scale design optimizations of electric vehicles have been limited in the past due to a lack of computational power. In order to fully utilize the potential of EV technology, novel optimization methods need to be developed. This dissertation investigates new opportunities for EV system optimization evolving from recent advancements in parallel processing hardware. The power train and control system of a fuel cell hybrid electric vehicle are optimized on the parallel architecture of graphics processing units. A two-level optimization methodology is developed specifically for peak performance on GPU architectures. Computational speedup factors of more than 2,000x and 70,000x are achieved over a sequential C/C++ implementation and the Matlab/Simulink environment, respectively. The significant gain in computation speed enables incorporation of expensive lifetime effects, such as battery degradation, as well as parameter uncertainties, such as variations in driving patterns. To achieve this, the entire lifetime of a vehicle is simulated during the optimization. Therefore, a novel methodology is proposed to generate stochastic drive cycles based on application and driver-specific driving profiles. Results show that the generated drive cycles accurately represent driving profiles with respect to the frequency spectra, speed distribution, acceleration distribution, and load characteristics of their corresponding duty cycles. Overall, the novel approach broadens the scope of conventional EV optimization methodologies and therefore increases the significance of results in terms of quality and accuracy. Seizing this opportunity, a variety of studies are performed that reveal significant fuel efficiency gains through driver and application-specific lifetime optimizations.

Table of Contents

List of Tables

List of Figures

List of Symbols

A_{FS} Amplitude

A_{front} Front area

E Ideal electrode potential

E_0 Battery constant voltage

$E_{bat_{max}}$ Maximum energy content of battery

F_{acc} Aerodynamic drag

F_{drag} Rolling resistance

F_{grav} Inertia force for acceleration

F_{roll} Projected normal force

F_{total} Total force on vehicle

G Gibbs free energy

I_{fc} Fuel cell current

I_{st} Stack current

K Polarization constant

$P(x)$ Cumulative probability distribution

P_{bat}^{+} Discharging power demand to battery pack

P_{bt} Power demand to battery pack

P_{fc} Power demand to fuel cell stack

P_{mt} Rated motor power

Q Battery capacity

Q_c Actual battery charge

R Gas constant

R_{int} Internal Resistance

S_0 Standard molar entropy

T_0 Standard temperature

W_{el} Maximum obtainable work from an electrochemical reaction

a Vehicle acceleration

c_d Drag coefficient

d_{lt} Lifetime mileage of a vehicle

erf Gauss error function

$f(x)$ Fitness function

f_i Fugacity of species i

g Gravitational acceleration

i Number of population members of outer loop

i^* Filtered current

i_l Limiting current density

i_{bt} Battery current

j Number of population members of inner loop

k Number of drive cycles in the lifetime of the vehicle

m Vehicle mass

m_{H_2} Overall hydrogen consumption

m_{bt} Weight of battery pack

$m_{f_{H_2}}$ Hydrogen consumption for one drive cycle

m_{fc} Weight of fuel cell system

m_{mt} Weight of motor

p Component design vector

q EMC parameter design vector

s Scaling coefficient

t_{DC} Duration of drive cycle

t_{DP} Duration of driving pulse

t_m Thickness of the electrolyte membrane

v Velocity of the vehicle

v_{act} Activation Loss

v_{conc} Concentration Loss

v_{ohm} Ohmic Loss

v_{st} Overall stack voltage

v_{st} Overall stack voltage

x Design vector

ΔG° Gibbs free energy change at standard pressure

ΔP Maximum power intensity in lifetime of vehicle

$\Phi(x)$ Cumulative distribution function

α_1 EMC adjustment parameter

α_2 EMC adjustment parameter

$\dot{m}_{H_2}$ Hydrogen consumption rate

γ EMC adjustment parameter

λ_m Water content of membrane

$\mathbb{S}$ Search Space

μ Tire rolling resistance

ω_{FS} Frequency

ρ_{air} Density of air

σ_m Membrane conductivity

θ Road angle

φ_{FS} Phase

List of Abbreviations

ACO Ant Colony Optimization
API Application Programming Interface

BEV Battery Electric Vehicle

CMR Candidate Mutation Rate
CUDA Compute Unified Device Architecture

DC Drive Cycle
DCGT Drive Cycle Generation Tool
DOH Degree of Hyridization
DP Driving Pulse
DS Driving Scenarios
DyP Dynamic Programming

ECMS Equivalent Consumption minimization strategy
EM Energy Management
EMC Energy Management Controller
EMS Energy Management Strategy
EV Electric Vehicle

FCHEV Fuel Cell Hybrid Electric Vehicle
FPS Fitness Proportionate Selection

GA Genetic Algorithm
GLSL OpenGL Shading Language
GPGPGPU Genetic Programming on General Purpose Graphics Processing Units
GPGPU General Purpose Computing on Graphics Processing Units
GPU Graphics Processing Unit

HEV Hybrid Electric Vehicle
HF High frequency
HIG Highway
HLSL High-Level Shading Language

IC Internal Combustion
ICE Internal Combustion Engine

LF Low frequency

MF Medium frequency

NLLSR Non-linear least square regression

ODE Ordinary Differential Equation
OETS Odd-Even Tranposition Sort

PdPFD Power - delta Power Frequency Distribution
PEMFC Proton Exchange Membrane Fuel Cell
PHEV Plug-in Hybrid Electric Vehicle
PMR Parameter Mutation Rate

RUR Rural

SA Simulated Annealing
SAFD Speed-Acceleration Frequency Distribution
SDP Stochastic Dynamic Programming
SIMD Single Instruction Multiple Data
SM Streaming Multiprocessor
SNG Stop-And-Go
SOC State of Charge
SQP Sequential Quadratic Programming
SUB Suburban

UAV Unmanned Aerial Vehicle
URB Urban

Chapter 1

Introduction

1.1 Problem Description and Scope of the Research

Motivation

Considering the rising cost of gasoline, political dependencies caused by resource imports and tougher fuel-efficiency standards, electric vehicles (EVs) and hybrid electric vehicles (HEVs) appear to be a worthwhile alternative to internal combustion (IC) power trains. This is especially true when the vehicles are powered by low and zero-emission renewable energy sources. As a result, the transition from internal combustion engine (ICE) cars to EVs is becoming desirable for consumers. The market demand for EVs is predicted to significantly increase in the next decade. However, only recently have car manufacturers started to push EVs into the consumer automotive market. While ICE technology has enjoyed the benefits of over 130 years of research and the growth of a technology-specific infrastructure, modern EVs are far from being technologically matured. These shortfalls need to be made up in order to be competitive in the automotive market as EVs are expected to fulfill similar technical standards and system characteristics that are found in ICE vehicles. EVs have the potential to be very energy efficient due to the interconnection of power sources with different energetic properties. Most EV concepts such as hybrid electric vehicles, plug-in hybrid electric vehicles (PHEV) or fuel cell hybrid electric vehicles (FCHEV) are characterized by a high degree of system complexity when compared with conventional ICE power trains. The complexity results from the combination of different power sources, electrical and mechanical components and a sophisticated energy management strategy (EMS). Additionally, EV power sources suffer from degradation. The power output capabilities of batteries and fuel cells change over the course of the vehicle's lifetime depending on their utilization. Moreover, varying degradation rates of power sources might cause a continuous shift of the degree of hybridization (DOH). This transient sys-

tem state needs to be considered in the conceptual design phase of the vehicle. In short, EV power train designs are significantly different from conventional ICE designs. Traditional ICE design optimization methodologies will therefore not carry over to EV applications without modifications. Consequently, there is a need for novel optimization methodologies specifically tailored to the demands of EVs.

Previous Work concerning Optimization of Energy Management Controller

A large number of studies dealing with EV optimization have been published, which mostly focus on energy management optimization. State-of-the-art control strategies for HEVs were classified and reviewed by Salmasi [19]. Won and Langari [20] studied a fuzzy logic-based control approach for hybrid vehicles. The proposed control system was evaluated under the FTP75[1] urban drive cycle. Johannesson and Egardt [21] employed dynamic programming (DyP) methodologies to determine the optimal control trajectories for HEVs, which can then be used as a benchmark to design control rules. Linear approximations of the value function and model approximations were utilized to reduce the computation time. Lin et al. [24] applied the DyP technique to solve the optimal power management problem of a hybrid electric truck by minimizing a cost function over one drive cycle. Simplifications and look-up tables were used to achieve a reasonable computation time. Furthermore, Lin et al. [25] investigated using stochastic dynamic programming (SDP) to find the optimal control strategy for HEVs. The power demand from the driver was modeled as a random Markov process. A limited number of state variables was kept in order to minimize computation time. Rodatz et al. [22] determined the real-time optimal power distribution of a fuel cell powered vehicle using the equivalent consumption minimization strategy (ECMS). The proposed methodology was tested with the NEDC[2]. Tulupe et al. [23] benchmarked a ECMS power management against globally optimal dynamic programming.

Previous Work concerning Optimization of Component Design

Various approaches have also been proposed for component design optimization. Fellini et al. [28] provide an overview of various software design environments and optimization algorithms for HEV design optimization. Kim and Peng [29] developed a combined power management and design optimization framework and tested it for 3 different drive cycles. The vehicle simulation was simplified to make it suitable for the computationally expensive optimization process. Assanis

[1] The **F**ederal **T**est **P**rocedure FTP75 is a drive cycle established by the United States Environmental Protection Agency for emission testing.

[2] The **N**ew **E**uropean **D**riving **C**ycle (NEDC) was designed to represent typical European city driving.

et al. [30] optimized the fuel economy of an HEV in the ADVISOR environment using sequential quadratic programming (SQP). The crucial relationship between accuracy and computation time is discussed in this paper. Gao and Porandia [32] compared DIRECT, a genetic algorithm and simulated annealing for optimizing a parallel hybrid electric power train design using a combined FTP-75/HWFET[3] drive cycle. Optimization times of more than 100 hours are reported caused by the slow evaluation speed of PSAT[4]. Zhang et al. [31] proposed a multi-objective parameter optimization for a series hybrid electric vehicle and tested it with a combined UDDS[5] /HWFET drive cycle. Liu et al. [34] developed a hybrid genetic algorithm for optimal component sizing and studied its performance for the City-Hwy test procedure in the ADVISOR[6] environment. Allison et al. [26, 27, 4] studied the effect of optimal partitioning and decomposition-based design optimization for complex engineering systems, such as HEVs. The goal is to achieve faster optimization times while maintaining similar accuracy.

Increasing Computation Speed

Looking at the above references, it becomes clear that most EV optimization engineers face similar challenges, which are very slow overall computation times and the resulting need for simplifications. Software simulation is an integral part of system design optimization. Physical simulation of EV power trains is computationally expensive. The computation time for large-scale design optimizations increases roughly proportionally to the number of simulations performed. For multi-parameter optimizations the computation time quickly becomes infeasible since the number of required simulations grows exponentially with the number of state variables. The following measurements are identified from the cited references as possible solutions to reduce optimization times:

1. Simplifying the objective function (system simulation):

    - Minimizing the number of state variables reduces the dimensionality of the simulation.
    - Replacing computationally expensive operations with approximation functions or look-up tables saves processing time by substituting complex runtime computations with simpler (array indexing) operations.

[3]The Highway Fuel Economy Cycle (HWFET) is a chassis dynamometer drive cycle developed by the United States Environmental Protection Agency for the determination of fuel economy of light duty vehicles.

[4]PSAT is a **P**owertrain **S**ystem **A**nalysis **T**oolkit developed by the Argonnen National Laboratory.

[5]The **U**rban **D**ynamometer **D**riving **S**chedule is a dynamometer test on fuel economy that represents city driving conditions. It was developed by the United States Environmental Protection Agency.

[6]ADVISOR is a Matlab-based **Ad**vanced **V**ehicle **S**imulat**or** developed by the National Renewable Energy Laboratory.

- Reducing the order of numerical ordinary differential equation (ODE) solvers decreases the computation time.
- Increasing the simulation stepsize proportionally decreases the number of processor operations, if the system is modeled in a discrete fashion.

The trade-off for all of the above measurements is a loss of accuracy.

2. Simplifying the objective:

   - In case of a large input dataset, smaller representative inputs can be designed based on the stochastic characteristics of the original set. For example, one average-assumed drive cycle can be employed instead of an entire lifetime driving profile.
   - For multi-objective optimization, reducing the number of objectives exponentially decreases the complexity of the optimization problem.

   The trade-off for both measurements is the inability to account for all stochastic eventualities (lifetime effects, uncertainties), and the resulting loss of accuracy and result variety.

3. Selecting the most suitable optimization algorithms and implementing a task-specific optimization framework:

   - Determining the optimal parameter setup for each specific problem improves the efficiency of the optimization algorithm.
   - System decomposition and partitioning are efficient tools to improve the optimization time.

   Besides an increased implementation effort, other trade-offs for the proposed measurements are negligible.

There are two ways to improve the speed of a computer-based optimization process: Improving the software or accelerating the computation hardware. We have seen that a variety of options have been proposed in the literature for improving computation times in terms of software modifications. On the other hand, very little has been reported on the effect of computation hardware on the optimization time. The main scope of this dissertation is to investigate the potential benefit of employing modern computation hardware for large-scale design optimizations.

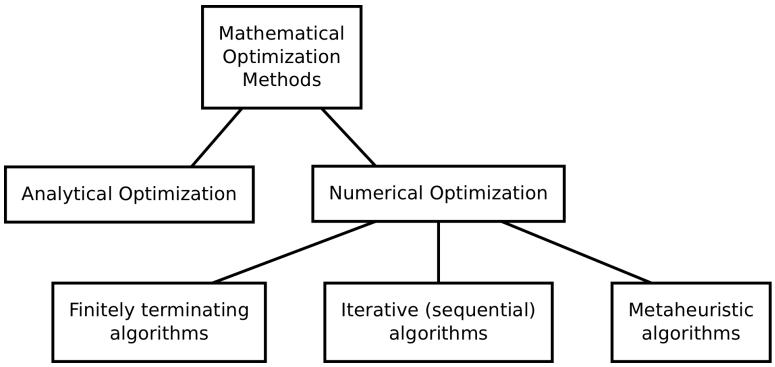
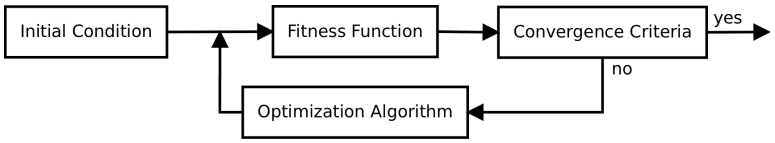
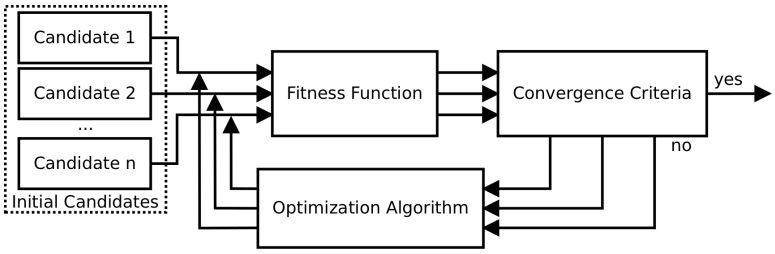


Figure 1.1. A schematic classification of mathematical optimization methods



(a) Iterative optimization routine



(b) Meta-heuristic optimization routine

Figure 1.2. Schematic diagram of a typical iterative and meta-heuristic optimization method

Sequential versus Parallel Computing for Optimization

To understand the importance of computation hardware on optimization algorithms, it is necessary to look at how different algorithms utilize the hardware. Fig 1.1 illustrates a schematic classification of mathematical optimization methods that can be employed in engineering optimization. On the top level, there are derivative-based analytical algorithms and numerical algorithms. The latter are traditionally used for non-linear problems that are too complex to determine an analytical solution. Numerical optimization is classified into *finitely terminating algorithms, iterative or sequential algorithms* and *meta-heuristic algorithms*. A typical example of finitely terminating algorithms is the *simplex algorithm*[76]. Examples of iterative algorithms are *Newton's method* [78], *sequential quadratic programming* [79], and *gradient descent* [77]. Finally, examples of meta-heuristic algo-

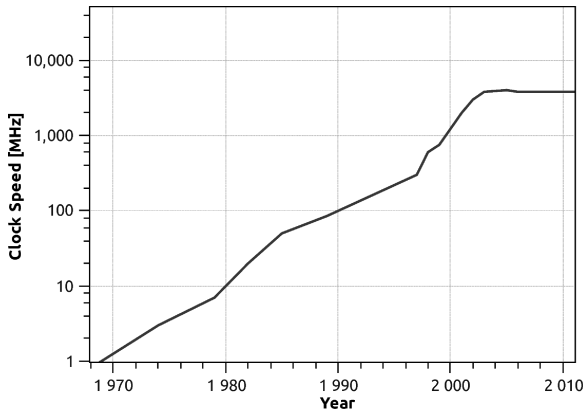


Figure 1.3. Development of the processor clock speed since 1969. Since 2003, the clock speed has not increased due to thermodynamic limitations.

rithms are *genetic algorithms (GAs)* [69, 70, 13], and *simulated annealing (SA)* [71, 72, 73] *ant colony optimization (ACO)* [74, 75]. Most iterative algorithms are comparatively easy to implement and guarantee at least local convergence. The implementation of meta-heuristic algorithms is somewhat more challenging. In most cases, neither global nor local convergence can be guaranteed. Fig. 1.2(a) illustrates the general principle of iterative optimization routines. The inherently serial properties of the process are recognizable. Each new iteration depends on the results of the previous iteration until convergence is achieved. Most meta-heuristic algorithm have a similar outer setup. However, the key difference is that meta-heuristic algorithms are based on a set of possible solution candidates while iterative algorithms operate with just one candidate (Fig. 1.2(b)). The computationally expensive evaluation of the candidate's fitness can be parallelized for meta-heuristic algorithms under normal circumstances. Furthermore, it is possible to program certain operations of meta-heuristic optimization routines in parallel, such as parameter sorting.

Typically, design engineers have limited access to supercomputers and large-scale CPU clusters. In addition, parallel implementations of system models and optimization routines are significantly more challenging and time-consuming than sequential implementations. As a consequence, most optimization algorithms used in engineering design are inherently sequential or are meta-heuristc algorithms implemented in a serial fashion. However, inherently sequential optimization algorithms are directly dependent on the clock speed of the CPU since they are single-threaded. According to Moore's Law, the number of transistors on integrated circuits doubles every 18-24months [68]. In the past, Moore's Law had been linked to the clock speed of CPUs. However, the clock speed of CPUs has not been notably increased since 2003 due to thermodynamic

limitations. The development of clock speed since 1969 is illustrated in Fig. 1.3. Nevertheless, Moore's Law is still valid as the transistor rate is still increasing through the integration of multiple computation cores into a single processing unit. As a consequence of this development, modern computation units will not become significantly faster in the future, but instead will feature increased multi-threading capabilities. Therefore, inherently sequential optimization algorithms will not benefit from future improvements in computation hardware, in fact, they have already peaked in terms of hardware acceleration. On the other hand, meta-heuristic search algorithms are parallelizable for the most part, allowing them to take full advantage of multi-threaded hardware. Optimization engineers that adapt to this development will benefit from both drastically reduced computation times and the ability to solve optimization problems found previously infeasible on conventional architectures.

Potential of Graphics Processing Units

In recent years, the parallel computing architecture of graphic processing units (GPU) has evolved more significantly than any other parallel processing architecture, such as modern multi-core CPUs. GPUs have become increasingly parallel to accomplish advanced 3D graphic rendering techniques for modern gaming software. In their beginning, GPUs were only programmable with special graphics application programming interfaces (APIs). With the OpenGL[7] API and Microsoft's DirectX[8] API, flexible higher level shader programming was made possible through the High-Level Shading Language (HLSL) and the OpenGL Shading Language (GLSL), respectively [82]. Shaders are software instructions that were primarily used to calculate rendering effects on GPUs. HLSL was designed for three types of shaders: vertex shaders, geometry shaders and pixel shaders. The parallalizable properties of shading explain the fast evolution of the parallel processing architecture of GPUs. However, general purpose programming used to be very restricted since computations had to be expressed in graphical terms like vertices, textures, fragments, and blending. In 2007, the California-based company NVidia introduced CUDA (Compute Unified Device Architecture), a parallel computing architecture, alongside with C for CUDA, a C-based API for GPU programming. The extremely high floating point performance of consumer low-cost GPUs, together with an improved programming interface, allows for the execution of scientific applications on desktop computers [82]. Nowadays, modern GPUs are not only graphics engines, but also highly parallel processors that significantly outpace CPUs for certain applications. Since most design engineers

[7] **Open G**raphics **L**ibrary (OpenGL) is a cross-platform API for writing programs that produce computer graphics.
[8] DirectX is a collection of APIs for handling multimedia tasks, such as gaming and video.

have no access to supercomputers, GPU computing is becoming a powerful and affordable alternative. The sequential part of the application runs on the CPU and the computationally-intensive part is accelerated by the parallel GPU [81]. The hardware architecture of a modern GPU follows a single instruction multiple data (SIMD) programming model. Ideally, a GPU processes many threads in parallel for the same instructions. Due to the complex memory architecture of GPUs, inter-thread communication is possible but not trivial to implement. The technique of using GPUs to perform computations that are typically executed on CPUs is called general-purpose computing on graphics processing units (GPGPU) [83, 84]. Researchers in various fields have reported significant computational speedups by employing GPGPU techniques. Thibault et al. [85] reported two orders of magnitude of speedup relative to a serial CPU implementation of a Navier-Stokes solver for incompressible flows on a multi-GPU desktop. Stivala et al. [86] achieved a 34x speedup over a CPU implementation of a tableau-based protein substructure search with simulated annealing. Komatitsch et al. [87] achieved a speedup of up to 20x over a reference CPU cluster for a high-order finite-element application, which performed the numerical simulation of seismic wave propagation on a large cluster of NVidia Tesla graphics cards. Geveler et al. [88] measured an eightfold speedup using modern GPUs in contrast to multi-threaded CPU code for the simulation of laminar fluid flows based on the two-dimensional shallow water equation and the Lattice-Blotzmann method. Other research areas that have benefited from GPU computing include: data mining, computational finance, medical imaging, and more.

Previous Work concerning Optimization on GPUs

Promising results of GA implementations on GPUs have also been achieved within the computer science community. The research field is called Genetic Programming on General Purpose Graphics Processing Units (GPGPGPU). So far, the majority of the publications focus solely on the hardware-specific implementation of the algorithms and their performance compared to traditional CPU implementations. Harding et al. [89, 90] demonstrated that the use of GPUs accelerates evolutionary computation applications by up to several hundred times over typical CPU implementation. Simple benchmark functions were used as fitness functions. Fok et al. [91] were able to show speedups of up to 5x when they ported evolutionary programming to a GPU. The operators of mutation, selection, and fitness calculation were fully implemented on their GPU. Harding and Banzhaf [92] proved that splitting the program compilation, fitness case data, and fitness execution over a cluster of GPU equipped computers further improves computation speeds. Pospichal et al. [93] mapped a parallel island-based GA with unidirectional ring migrations to a CUDA software

model. Tests were performed using Rosenbrocks, Griewanks and Michalewiczs benchmark functions. The proposed approach led to speedups of up to seven thousand times higher compared to one CPU thread. Furthermore, Pospichal et al. [95] showed that a consumer-level GPU can be used to significantly speed up optimization of the Knapsack[9] problems. Speedups of up to 1340x were reported, however, Pospichal also elaborated on the limitations of GPU computing for evolutionary computations. The main finding was that GPUs must be utilized with sufficient block and thread sizes in order to overcome the disadvantage of memory latency. Sato et al. [94] demonstrated the possibility of solving Sudoku puzzles on GPUs while achieving a practical processing time. The achieved speedup was reported to be up 25x over a single-threaded CPU implementation in C/C++ and up to 96x over a single-threaded Java implementation. Luong et al. [96] used the GA island model to solve large-scale and time-intensive combinatorial optimization problems with GPUs. Speedups of over 2000x were achieved in this study.

Scope of this Dissertation

Using GPGPGPU for a large-scale design optimization of full engineering systems such as EVs is a novel approach. While it has been shown that GAs for simple benchmark functions can efficiently be implemented on GPUs, parallel simulation of engineering systems as part of an objective function is a more challenging task. The scope of this research is to investigate the potential speedup of performing a large-scale design optimization of a FCHEV on GPUs. For speed benchmarking, a backward-looking system simulation is created in three different programming environments: CUDA for GPU programming, the intermediate-level programming language C/C++, and the graphical programming environment Matlab/Simulink. The CUDA code is then mapped into the fitness function of a GPU-based GA optimization routine that is specifically designed for the optimization of the FCHEV. The proposed optimization approach is mostly generic and can easily be employed to other transient-state simulations of engineering systems with only minor modifications. The resulting speedup of this novel implementation concept compared to the C/C++ and Matlab/Simulink implementations is studied and utilized to improve conventional EV optimization methodologies. As mentioned before, single-threaded CPU optimizations of EV systems are, in many cases, computationally infeasible without simplifications, resulting in a loss of accuracy and additional constraints on the research scope. Fast GPU-based design optimization is not constrained by the multitude of simplifications that are commonly used to avoid impracticable computation times on single-threaded

[9]The Knapsack problem is a problem in combinatorial optimization, which is often used as a benchmark problem.

CPU systems. Thus, GPU-based design optimization provides new opportunities to study the optimal system design of EVs and gain a thorough understanding of key design challenges.

Stochastic simplification of optimization input parameters has been identified as a prevalent procedure. It is common practice to use just one drive cycle, for example UDDS, which is designed as a stochastic representation of an entire driving profile, as input data for the objective function. Driving profiles are the collection of all drive cycles in the lifetime of a vehicle. Evidently, performing a vehicle simulation for a driving profile ($\sim$100,000-200,000km) is much more computationally expensive than a simulation for just one drive cycle ($\sim$10-100km). GPU-based EV optimization allows for the utilization of entire driving profiles in the objective function, thus enabling the extension of conventional optimization strategies with the following prospects:

1. Consideration of lifetime effects: Incorporating degradation effects into the simulation allows for the accounting of the transient state of components, such as battery packs and fuel cell stacks.

2. Consideration of parameter and model uncertainties: Driving profiles automatically account for most variations of driving patterns. Parameter uncertainties can be represented by probability functions along with random number generators, since the driving profile simulation space is large enough for an accurate stochastic representation.

3. Vehicle design individualization: Driving profiles can be individually created according to the driving characteristics of a specific driver. The resulting optimization would provide the optimal car for the needs of this driver.

4. Sensitivity studies: The effects of parameter sensitivities can be thoroughly investigated by executing multiple optimization runs. Small changes of model parameters or input parameters might have a considerable effect on the overall system design.

1.2 Dissertation Outline

Fig. 1.4 illustrates the outline of this dissertation. Chapter 1 introduces the scope of this research. It presents an overview of previous work and discusses the motivation and contributions of this dissertation. Chapter 2 describes the physics of the FCHEV model and its implementation into three different programming environments. Chapter 3 proposes a novel methodology to generate stochastic drive cycles based on application and driver-specific driving profiles. Chapter 4

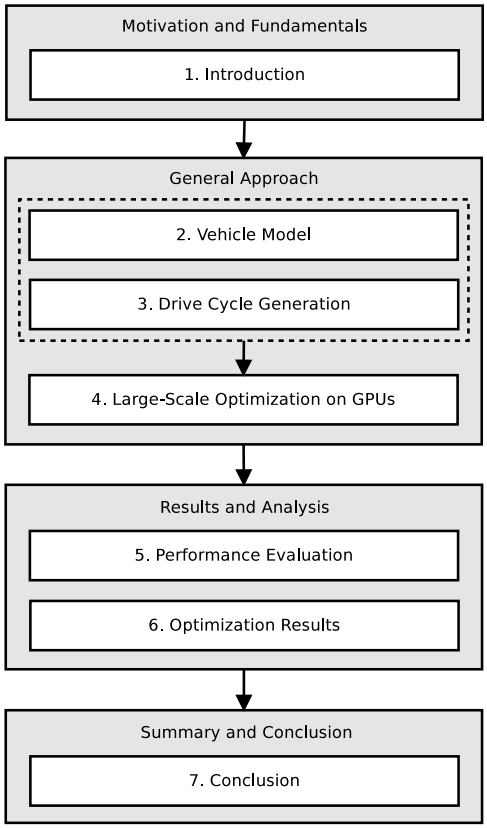


Figure 1.4. Dissertation Outline

incorporates the vehicle model and drive cycle generation tool into a two-level optimization routine which is specifically designed for GPUs. The performance of the proposed optimization methodology is evaluated and analyzed in Chapter 5.6. Results of various FCHEV optimization scenarios are presented and discussed in Chapter 6. Chapter 7 summarizes this dissertation, emphasizes the conclusions of this research, and proposes future work.

Chapter 2

Vehicle Model

A discrete backward-looking system simulation of a FCHEV is created in three different programming environments: CUDA for GPU programming, C/C++, and Matlab/Simulink. Figure 2.1 illustrates the system architecture of the vehicle.

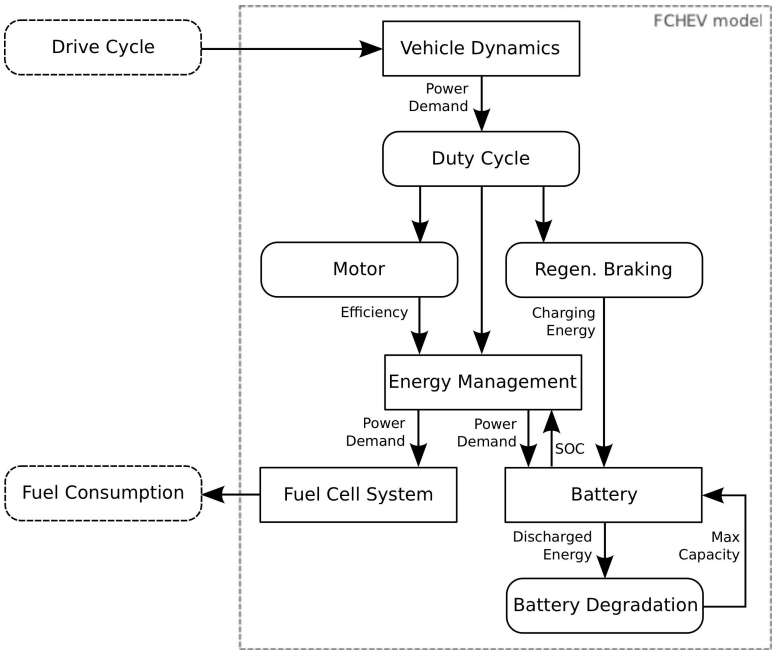


Figure 2.1. System architecture and EMS setup of FCHEV model consisting of fuel cell stack and battery pack as energy buffer. The model incorporates the effects of regenerative brake energy recovery and battery degradation.

A drive cycle is the input to the model. Drive cycles are represented as an array of vehicle speed data and possibly elevation data for each discrete timestep. The vehicle dynamics model converts the drive cycle into a duty cycle considering the physical properties of the vehicle. Duty

Parameter	**Notation**	**Value**
Base vehicle weight [1]	m_{bw}	500kg
Fuel cell type	$-$	PEMFC
Battery type	$-$	Lithium-Ion
Brake recuperation efficiency	μ_{btr}	30%
Timestep size	t_s	0.2s
Drag coefficient	c_d	0.26
Tire rolling resistance	μ	0.013
Front area	A_{front}	$2m^2$
Active area of fuel cell	A_{fc}	$200cm^2$
Fuel cell membrane thickness	d_m	1.25mm
Water content of membrane	λ_m	14 ($\equiv$100%)

Table 2.1. Specifications of the FCHEV system model

cycles are arrays of vehicle power demand data, consisting of one data point for each discrete timestep. The vehicle is powered by a proton exchange membrane fuel cell (PEMFC) stack and a lithium-ion battery pack as an energy buffer. The battery pack is either charged by excess energy provided from the fuel cell stack or by regenerative braking with a recuperation efficiency of 30%. A component degradation model keeps track of the battery utilization and computes the resulting loss of capacity while the energy management controller (EMC) manages the power flows between the two power sources. A detailed description of the EMC can be found in Section 2.5 and Table 2.1 provides specifications for the vehicle. Each system component is seperately modeled obeying the principles of conservation: mass, momentum, species, charge and thermal energy. The component models are then interconnected to create the overall system simulation.

2.1 Vehicle Dynamics Model

The power demand of the vehicle is estimated by a vehicle dynamics model. The model calculates the incremental change in position of the vehicle and the external forces for each iteration of the simulation. The external forces are inertia force for acceleration (F_{acc}), projected normal force (F_{grav}), rolling resistance (F_{roll}) and aerodynamic drag (F_{drag}) [33].

[1] Vehicle weight without fuel cell system, battery pack and motor

$$F_{acc}(t) = m \cdot a(t) \tag{2.1.1}$$

$$F_{grav}(t) = m \cdot g \cdot \sin(\theta(t)) \tag{2.1.2}$$

$$F_{roll}(t) = \mu \cdot m \cdot g \cdot \cos(\theta(t)) \tag{2.1.3}$$

$$F_{drag}(t) = \frac{1}{2} \cdot \rho_{air} \cdot v(t)^2 \cdot c_d \cdot A_{front} \tag{2.1.4}$$

$$F_{total}(t) = F_{acc}(t) + F_{grav}(t) + F_{roll}(t) + F_{drag}(t) \tag{2.1.5}$$

Here, m is the vehicle mass, a is the acceleration, g is the gravitational acceleration, θ is the road angle, μ is the rolling resistance coefficient, ρ_{air} is the density of air, v is the velocity of the vehicle, c_d is the drag coefficient and A_{front} is the vehicle's front area. The total force F_{total} on the vehicle is the sum of all external forces. In a discrete notation, Eqn. 2.1.1-2.1.5 can be expressed as:

$$F_{acc_n} = m \cdot \frac{v_n - v_{n-1}}{t_s} \tag{2.1.6}$$

$$F_{grav_n} = m \cdot g \cdot \sin\left(\arctan\frac{h_n - h_{n-1}}{t_s \cdot v_n}\right), v_n \neq 0 \tag{2.1.7}$$

$$F_{roll_n} = \mu \cdot m \cdot g \cdot \cos\left(\arctan\frac{h_n - h_{n-1}}{t_s \cdot v_n}\right), v_n \neq 0 \tag{2.1.8}$$

$$F_{drag_n} = \frac{1}{2} \cdot \rho_{air} \cdot v_n^2 \cdot c_d \cdot A_{front} \tag{2.1.9}$$

$$F_{total_n} = F_{acc_n} + F_{grav_n} + F_{roll_n} + F_{drag_n} \tag{2.1.10}$$

where h is the elevation of the vehicle. Due to analytical restrictions, $v_n = 0$ has to be replaced with $v_n \to 0+$. The overall power demand is obtained with:

$$P_{total_n} = F_{total_n} \cdot v_n \tag{2.1.11}$$

2.1.1 Component Weight Functions

The hypothetical component weight functions of the motor m_{mt}, fuel cell system m_{fc} and battery pack m_{bt} are illustrated in Fig. 2.2 and represented by the following equations:

$$m_{mt} = (-0.007756 \cdot |P_{max}|^2 + 5.6 \cdot |P_{max}| + 26.88)kg \tag{2.1.12}$$

$$m_{fc} = (80 + n_{fc}/2)kg \tag{2.1.13}$$

$$m_{bt} = (50 + C_{max}/200)kg \tag{2.1.14}$$

The overall vehicle weight is computed as:

$$m = m_{bw} + m_{mt} + m_{fc} + m_{bt} \tag{2.1.15}$$

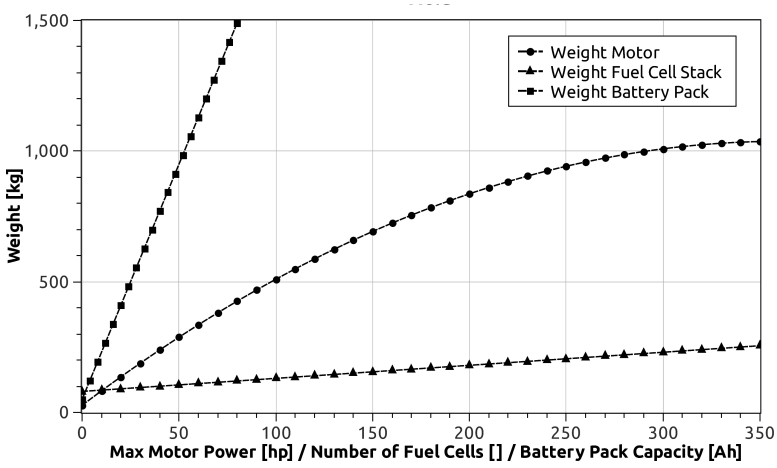


Figure 2.2. Hypothetical component weight functions of motor, fuel cell system, and battery pack.

2.2 Fuel Cell Stack Model

Fuel cells convert chemical energy contained in a fuel, such as hydrogen, into electrical energy. A typical fuel cell consists of an anode and a cathode on either side, and an electrolyte membrane in between. A schematic block diagram of a fuel cell illustrating gas and ion flow directions is shown in Fig. 2.3 [49]. Hydrogen is continuously fed to the andode and oxygen to the cathode while electrons are drawn from the anode side and transported to the cathode side through an external circuit, thus performing work on the load. The overall reaction in a hydrogen/oxygen fuel cell can be written as:

$$H_{2(g)} + \frac{1}{2} O_{2(g)} \rightarrow H_2O_{(l)} \tag{2.2.1}$$

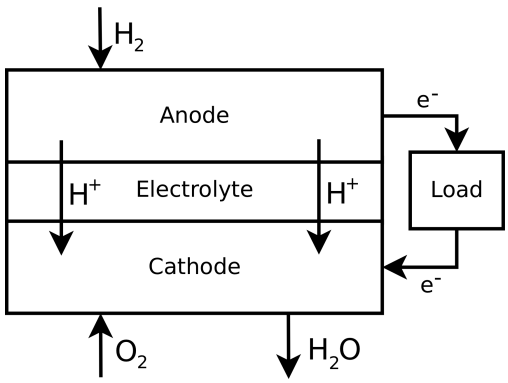


Figure 2.3. Block diagram of a fuel cell

2.2.1 Cell Voltage Model

The fuel cell voltage is computed as a function of pressure, temperature, reactant partial pressure, and relative humidity [43, 44, 45, 46, 47, 48, 49]. The maximum potential of a fuel cell can be obtained from the Gibbs free energy. The change in Gibbs free energy (ΔG) is the maximum obtainable work (W_{el}) from an electrochemical reaction at a constant temperature and pressure.

$$W_{el} = \Delta G = -nFE \tag{2.2.2}$$

It can also be expressed as:

$$\Delta G = \Delta G^{\circ} + RT \ln \left(\frac{f_C^{\gamma} f_D^{\delta}}{f_A^{\alpha} f_B^{\beta}} \right) \tag{2.2.3}$$

where n is the number of electrons per reacting ion or molecule, F is Faraday's constant (96,485 C/mol), E is the ideal electrode potential, ΔG° is the Gibbs free energy change of the reaction at standard pressure, R is the gas constant (8.3144 J/mol K), T is the temperature in Kelvin scale, and f_i is the fugacity of species i. Combining equations 2.2.2 and 2.2.3 leads to the Nernst equation:

$$E = E^{\circ} + \frac{RT}{nF} \ln \left(\frac{f_C^{\gamma} f_D^{\delta}}{f_A^{\alpha} f_B^{\beta}} \right) \tag{2.2.4}$$

Since PEMFCs typically operate at low pressures, the fugacity can be approximated by partial pressures.

$$E = E^{\circ} - \frac{RT}{nF} \ln \left(\frac{p_{H_2O}}{p_{H_2} \sqrt{p_{O_2}}} \right) \tag{2.2.5}$$

The standard state defines a standard state reference potential E_0^0, which is equal to 1.229V at 298.15K and 1atm. E^0 varies from the standard state reference in accordance with temperature [49] using:

$$E^0 = E_0^0 + (T - T_0) \left(\frac{\Delta S^0}{nF} \right) \tag{2.2.6}$$

with $E^0 = \beta_1 + \beta_2 T, \quad \beta_1 = 1.229V - \frac{298.15 \cdot \Delta S_0^0}{nF}, \quad \beta_2 = \frac{\Delta S_0^0}{nF}$

where T_0 is the standard temperature (298.15K) and S_0 is the standard molar entropy. The entropy change of a given reaction is approximately constant and can be set to the standard state value. According to Amphlett et al. [44], using literature values for the standard-state entropy change, the Nernst equation for a fuel cell can now be formulated as:

$$E = 1.229 - 0.85 \cdot 10^{-3} \cdot (T - 298.15) + 4.3085 \cdot 10^{-5} \cdot T \left[\ln p_{H2} + \frac{1}{2} \ln p_{O2} \right] \tag{2.2.7}$$

Voltage Losses

Several voltage losses occur in a fuel cell due to the combined effects of thermodynamics, mass transport, kinetics, and ohmic resistance. These physical and chemical factors determine the output voltage of the cell. The three major losses of a fuel cell are:

- Activation Overvoltage Loss or Activation Loss v_{act}
- Ohmic Resistance Loss or Ohmic Loss v_{ohm}
- Concentration Loss v_{conc}

Activation Loss Activation losses are caused by dissociation and ionization of gases in the electrodes. The activation overvoltage occurs at the anode as well as at the cathode, however, the anode loss can be neglected since it is very rapid. The relation between the activation overvoltage v_{act} and the current density is described by the Tafel equations [48]

$$v_{act} = a \cdot \ln \frac{i}{i_0} \tag{2.2.8}$$

where a is a constant and i_0 is the exchange current density. Since the Tafel equation is only valid for $i > i_0$, equation 2.2.8 can be approximated by the following equation for use in the fuel cell model [46].

$$v_{act} = v_0 + v_a(1 - e^{-c_1 i}) \tag{2.2.9}$$

where v_0 is the voltage drop at zero current density and v_a and c_1 are constants (see Appendix A).

Ohmic Loss The ohmic loss is caused by the resistance to the flow of ions in the membrane and the resistance to the flow of electrons through the electrodes. The voltage drop is proportional to the stack current

$$v_{ohm} = i \cdot R_{ohm} \tag{2.2.10}$$

with

$$R_{ohm} = \frac{t_m}{\sigma_m} \tag{2.2.11}$$

t_m is the thickness of the electrolyte membrane and σ_m is the membrane conductivity. According to Springer et al. [45], the membrane conductivity can be approximated with the following equation:

$$\sigma_m = (b_1 \lambda_m - b_2) \exp\left(b_3 \left(\frac{1}{303} - \frac{1}{T_{fc}}\right)\right) \tag{2.2.12}$$

where b_1, b_2 and b_3 are constants that have to be adjusted to fit the fuel cell data. The value of λ_m can vary between 0 (=0%) and 14 (=100%).

Concentration Loss Concentration losses or mass transport losses are caused from the change in concentration of the reactants as they are consumed in the reaction. Concentration losses are the reason for the rapid voltage drop at high current density and can be obtained with the following relationship:

$$v_{conc} = \frac{RT}{2F} \ln\left(1 - \frac{i}{i_l}\right) \tag{2.2.13}$$

i_l is a limiting current density at which the fuel is used up at a rate equal to its maximum supply rate [48]. Concentration losses are often ignored in fuel cell models since it is not desirable to operate the stack in regions where the concentration losses are very high.

2.2.2 Stack Model

The fuel cell operating voltage v_{fc} can be obtained by subtracting the voltage losses described in Section 2.2.1 from the open circuit voltage descirbed by Eqn. 2.2.7.

$$v_{fc} = E - v_{act} - v_{ohm} - v_{conc} \tag{2.2.14}$$

The overall stack voltage v_{st} is then obtained as the sum of the individual cell voltages:

$$v_{st} = \sum_{i=1}^{n_{fc}} v_{fc_n} \tag{2.2.15}$$

If all cells are identical, Eqn. 2.2.15 can be simplified to become:

$$v_{st} = n_{fc} \cdot v_{fc} \tag{2.2.16}$$

where v_{st} is the number of fuel cells in the stack. The stack current I_{st} is equal to the cell current I_{fc}. The current density is defined as the stack current per unit of cell active area:

$$i_{fc} = \frac{I_{st}}{A_{fc}} \tag{2.2.17}$$

The hydrogen consumption rate$\dot{m}_{H_2}$ is a function of the stack current using

$$\dot{m}_{H_2} = M_{H_2} \cdot \frac{n_{fc} I_{st}}{2F} \tag{2.2.18}$$

The overall hydrogen consumption for one drive cycle m_{H_2} is the integral of the hydrogen consumption rate over the duration of the drive cycle t_{DC}.

$$m_{H_2} = \int_{t=0}^{t=t_{DC}} \dot{m}_{H_2}(t) dt \tag{2.2.19}$$

In the FCHEV model, the overall fuel consumption for a drive cycle is obtained by converting the above equations to a discretized form. Using P_{fc} as the power demand to the fuel cell stack, the equations become:

$$i_{fc_n} = \frac{P_{fc_n}}{v_{fc_{n-1}} A_{fc}} \tag{2.2.20}$$

$$v_{fc_n} = E_n - v_{act_n} - v_{ohm_n} - v_{conc_n} \tag{2.2.21}$$

$$\Delta m_{fc_n} = m_{fc_n} - m_{fc_{n-1}} = M_{H_2} \cdot \frac{n_{fc}}{2F} \cdot \frac{P_{fc_n}}{v_{fc_n}} \cdot t_s \tag{2.2.22}$$

$$m_{H_2} = \sum_{n=0}^{k=\frac{t_{DC}}{t_s}} \Delta m_{fc_n} \tag{2.2.23}$$

2.3 Battery Model

A simple controlled voltage source in series with a constant resistance is used to model the lithium-ion battery. The actual state-of-charge (SOC) of the battery is the only state variable used to calculate the open source voltage with a non-linear equation. The following equations describes the battery voltage for charging:

$$v_{bt} = E_0 - R_{int} \cdot i_{bt} - K\frac{Q}{Q - Q_c} i^* - K\frac{Q}{Q - Q_c} Q_c + A \cdot e^{-B \cdot Q_c} \tag{2.3.1}$$

and for discharging:

$$v_{bt} = E_0 - R_{int} \cdot i_{bt} - K\frac{Q}{Q_c - 0.1 \cdot Q} i^* - K\frac{Q}{Q - Q_c} Q_c + A \cdot e^{-B \cdot Q_c} \tag{2.3.2}$$

where E_0 is the battery constant voltage, K is a polarization constant, i^* is the filtered current, ibt is the battery current, Q_c is the actual battery charge, Q is the battery capacity, R_{int} is the internal resistance, A is an exponential voltage, and B is an exponential capacity. A detailed description of the model can be found in references [50, 51].

The battery current is obtained from the power demand towards the battery P_{bt} and the battery voltage.

$$i_{bt} = \frac{P_{bt}}{v_{bt}} \tag{2.3.3}$$

The actual battery charge can be computed in a discretized form as follows:

$$Q_{c_n} = Q_{c_{n-1}} + i_{bt} \cdot t_s \tag{2.3.4}$$

The filtered current is approximated by:

$$i^*_n = (I_{bt} - i^*_{n-1}) \cdot (1 - e^{-0.2 \cdot t_s}) + i^*_{n-1} \tag{2.3.5}$$

Parameter	**Unit**	**Value**
Battery constant voltage E_0	[V]	3.366
Internal Resistance R	[Ω]	0.01
Polarization constant K	[Ω]	0.0076
Exponential voltage constant A	[V]	0.26422
Exponential capacity constant B	[$\frac{1}{Ah}$]	26.5487

Table 2.2. Battery Parameters

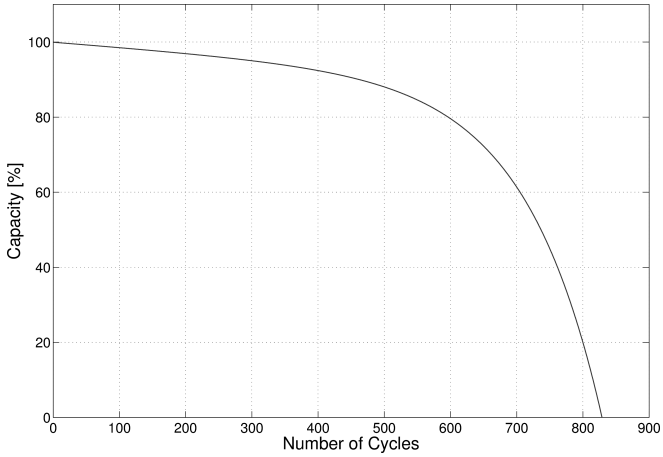


Figure 2.4. Hypothetical life-cycle curve of the lithium-ion battery pack for full charging cycles.

The battery SOC is defined as:

$$SOC = \frac{Q_c}{Q} \cdot 100 \qquad [\%] \tag{2.3.6}$$

The battery cell model is scaled up to pack level for the FCHEV simulation. Table 2.2 provides the specification parameters for the battery model.

2.3.1 Battery Degradation

Degradation of lithium-ion batteries is a complex process which depends on a large number of factors, such as depth of discharge, charge and discharge rates, temperature, etc. Various cycle life models have been published [52, 32, 54, 55, 59, 61, 65, 64, 63, 62]. However, the load on HEV battery packs compared to conventional battery testing methodologies distinguishes itself by dynamic, and rather short periods of charging and discharging caused by quick driving maneuvers and recuperative braking. There seems to be a lack of dynamic degradation models in the literature.

To simplify matters, a basic degradation model that is solely dependent on the energy usage of the battery pack is employed in this study. It can be easily replaced by more complex and accurate models when they areavailable. The proposed model is based on the hypothetical cycle life curve shown in Fig. 2.4. The cycle life curve is represented by a function $C(n)$, where n is the number of full charge-discharge cycles and $C(n)$ is the percent capacity of the battery with regard to the initial capacity. For the degradation model, n is obtained as

$$n = \frac{\int_0^t P_{bat}^+(t)}{E_{bat_{max}}} \tag{2.3.7}$$

P_{bat}^+ is the discharging power demand to the battery, t is the overall utilization time of the battery, and $E_{bat_{max}}$ is the maximum energy content of the battery, which is assumed to be constant in this study.

2.4 Motor Efficiency

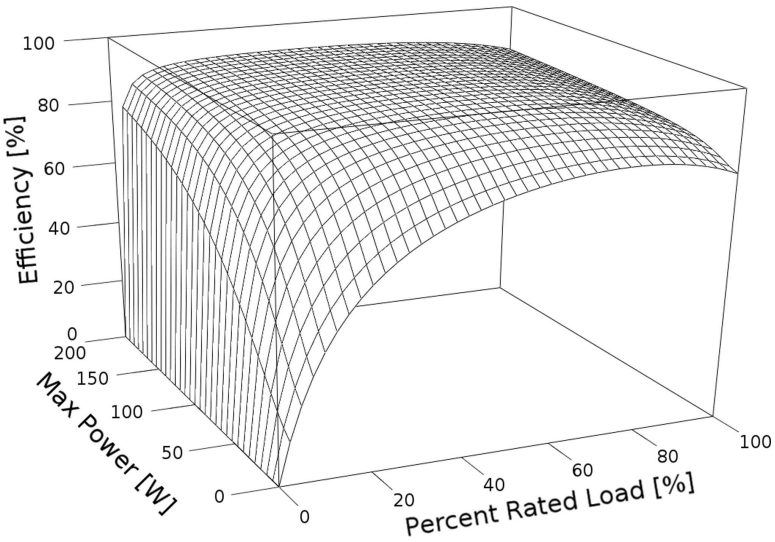


Figure 2.5. 3D Lookup table of the motor efficiency as a function of maximum rated power and power demand as a percentage of maximum rated load.

Electric motors are most efficiently operated at 50-100% of their rated load with the maximum efficiency usually close to 75%. Due to increased iron, stray, and mechanical losses, the motor efficiency rapidly drops if the motor is operated below 50% rated load. Fig. 2.5 illustrates a typical motor efficiency curve. The motor efficiency is ploted as a function of maximum rated power and

power demand as percentage of the maximum rated load. An analytic approximation of the pictured curve is implemented in the vehicle model.

2.5 Energy Management

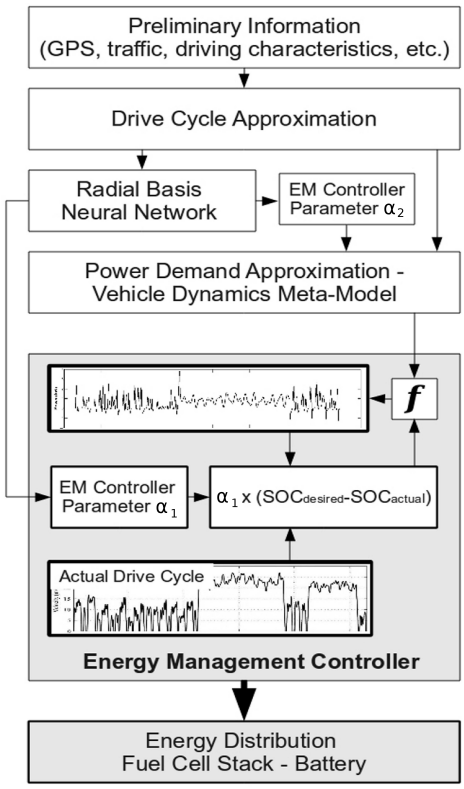


Figure 2.6. Energy Management Strategy for the FCHEV. The EMC leverages the SOC and power demand according to a precalculated driving scenario.

The main task of an energy management controller is to provide a near-optimal power split between the power sources in order to minimize fuel consumption. Since the proposed EMC is incorporated in an optimization routine of a backward-looking vehicle simulation, it needs to fulfill further requirements: First, the tuning of the EMC towards specific drive cycles should be performed using a limited number of controller parameters. Furthermore, the controller parameters should be easily optimizable towards the objective. A fast convergence time is desirable for the parameter optimization since it exponentially affects the overall optimization time of the vehicle (see Chapter 4).

A predictive two-parameter EMC is implemented in the FCHEV simulation. Fig. 2.6 illustrates the setup and functioning of the controller design. The controller uses preliminary driving data, such as GPS, terrain, and traffic data to generate an approximation of the expected drive cycle before operating the vehicle. The vehicle dynamics model utilizes this data to estimate the average power demand of the vehicle during driving scenarios. The average demand is then used as a baseline power demand for the fuel cell stack. The EMC employs two linear controller parameters. The first parameter α_1 levels the error between the average target SOC of a time period γ and the actual SOC of the battery pack. This can be viewed as a penalty function. The farther actual state is from the desired state, the stronger the impulse to reach the desired state. The second parameter α_2 adjusts the power demand approximation of the vehicle dynamics model (meta-model). The power demand adjustments are formulated as follows:

$$P_d(t) = [\Delta SOC(t,\gamma) \times (\alpha_1 + 1)] \times (\alpha_2 \times P_d^*(t)) \tag{2.5.1}$$

with

$$\Delta SOC(t,\gamma) = \overline{SOC_a(t,\gamma)} - SOC_d(t) \tag{2.5.2}$$

$$\overline{SOC_a(t,\gamma)} = \frac{1}{\gamma}\int_{\gamma}^{t^*} SOC_a(t)dt \tag{2.5.3}$$

$P_d(t)$ is the actual power demand, $P_d^*(t)$ is the predicted power-demand by the meta-model, $SOC_d(t)$ is the target state-of-charge, and $SOC_a(t,\gamma)$ is the actual state-of-charge of the battery pack. A value of γ=20 seconds has proven to be a good trade-off between not overstressing the dynamic capabilities of the fuel cell stack and achieving maximum fuel efficiency. $SOC_d(t)$ is set to a constant value of 50% in this study. The optimal controller parameters for each drive cycle are determined using an GA optimization routine. The GA finds the optimal controller settings that minimize the overall fuel consumption while maintaining a constant average SOC.

Since the proposed vehicle model is backward-looking and the entire simulation drive cycle is known in advance, the vehicle dynamics model can determine an accurate duty cycle before the execution of the actual simulation. This facilitates an increase in accuracy and a decrease in optimization time of the controller. References [36, 37] prove the applicability of the proposed concept to real-world driving scenarios. A radial basis neural network is used to determine the optimal controller parameters according to preliminary drive cycle data. The advantage of this approach is the fast execution time of neural networks compared to numerical optimization methods. Hence, the EMC is almost instantaneously operational while maintaining a very high degree of accuracy.

2.6 Programming Environments

The discrete FCHEV model is coded in three programming environments (CUDA, C/C++ and Matlab/Simulink) for speed benchmarking purposes.

2.6.1 CUDA

Algorithm 2.1 CUDA: n-parallel FCHEV simulations for 1 drive cycle

HOST:
init threadsize, blocksize
n = threadsize $\times$ blocksize
init DriveCycle, DesignParameter, Results, InitialConditions
cudaMalloc DriveCycle, DesignParameter, Results, InitialConditions
cudaMemcpyHostToDevice DriveCycle, DesignParameter, Results, InitialConditions
kernel <<blocksize, threadsize>> DriveCycle, DesignParameter, Results, InitialConditions
DEVICE:
init ModelParameter
for $i = 1 \rightarrow DriveCycleLength$ **do**
  VehicleDynamicsModel
  if $i = DriveCycleLength$ **then**
    return AveragePowerDemand
  end if
end for
for $i = 1 \rightarrow DriveCycleLength$ **do**
  VehicleDynamicsModel $\rightarrow$ EnergyManagementController $\rightarrow$ FuelCellModel $\rightarrow$ BatteryModel $\rightarrow$ Degradation-Model
  if $i = DriveCycleLength$ **then**
    return Results, InitialConditions
  end if
end for
HOST:
cudaMemcpyDeviceToHost Results, InitialConditions

C for CUDA is a high-level programming language based on C/C++, which facilitates co-processing on the CPU (host) and GPU (device). The entire vehicle simulation is embedded in a CUDA kernel. Drive cycle array data and design parameters are entirely mapped from the host memory to the shared memory of the GPU device, and constant variables are initialized on the device. Crucial state variables are also ported into the fast register memory of the GPU's multiprocessors. As a result, the FCHEV simulation for a full drive cycle is entirely executed on the

GPU, thus avoiding the large latency in CPU-GPU communication. Host-device memory transfer is usually the bottleneck for many GPU applications. To optimize performance, the simulation routine is therefore limited to one data transfer at the beginning of the simulation and one transfer at the end of the simulation.

Algorithm 2.1 illustrates the pseudocode of n-parallel CUDA simulations performed for one drive cycle. Drive cycle data and design parameters are initialized on the host and then allocated and transfered to the device memory. A kernel call starts the FCHEV simulation on the device. First, the vehicle dynamics model is executed to determine the average power demand for each driving scenario (see Section 2.5). Following this, the entire FCHEV simulation is performed utilizing the prior results to determine the optimal energy management configuration. The simulation results are then copied back and evaluated on the host.

2.6.2 C/C++

Algorithm 2.2 C/C++: 1 FCHEV simulations for 1 drive cycle

init DriveCycle, DesignParameter, Results, InitialConditions, ModelParameter
for $i = 1 \rightarrow DriveCycleLength$ **do**
  VehicleDynamicsModel
  if $i = DriveCycleLength$ **then**
    return AveragePowerDemand
  end if
end for
for $i = 1 \rightarrow DriveCycleLength$ **do**
  VehicleDynamicsModel $\rightarrow$ EnergyManagementController $\rightarrow$ FuelCellModel $\rightarrow$ BatteryModel $\rightarrow$ DegradationModel
  if $i = DriveCycleLength$ **then**
    return Results, InitialConditions
  end if
end for

Since the CUDA API is based on the C/C++ API, the structures of the CUDA code (Section 2.6.1) and the implemented C/C++ code are very similar. Algorithm 2.2 shows the pseudocode for one simulation performed on one CPU core. After the initialization of all parameters, the vehicle model routine is called to determine the average power demand for each driving scenario. The full FCHEV model is then executed and results are returned. The C/C++ implementation does not require advanced memory allocations and data transfers, since the CPU memory model is compara-

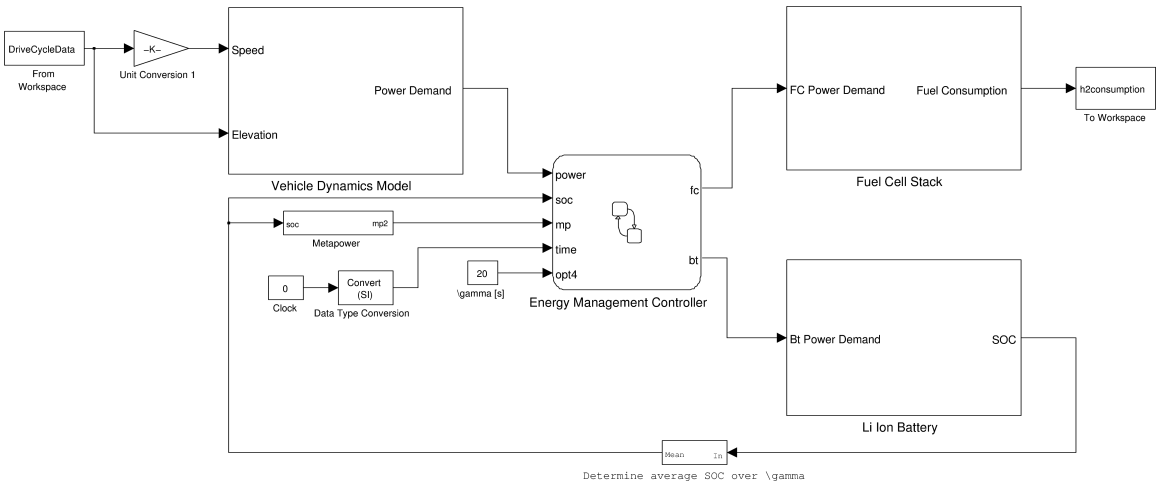


Figure 2.7. Simulink block diagram of the FCHEV model

tively simple. On the down side, the CPU routine is strictly sequential and does not provide parallel processing capabilities.

2.6.3 Matlab/Simulink

Simulink is a graphical multi-domain simulation and modeling tool for dynamic systems developed by the MathWorks corporation. To assure fair benchmarking, the FCHEV model is kept as simple and fast as possible. The model structure and math operations are analogically modeled after the CUDA and C/C++ routine. Time-consuming plotting and memory storage tools, as well as expensive Simulink library functions are not utilized. The Simulink-extension, Stateflow, is used as an environment for the EMC. Drive cycle data is loaded from an array in the Matlab workspace, and the simulation results are written to the workspace at the end of the simulation. The model and equation solvers are set as discrete using a fixed timestep-size of 0.2s. Fig. 2.7 illustrates the component interaction and data utilization of the FCHEV model in the Simulink environment.

Chapter 3

Drive Cycle Generation

Most of the proposed power management optimization methodologies and nearly all of the component design optimization methodologies presented in Section 1.1 share the fact that they use only one drive cycle in their objective function. This is a significant simplification from the ideal objective of evaluating a vehicle simulation for the entire lifetime of a vehicle. Instead, a smaller time range with compressed driving characteristics is used as input data. Sciaretta and Guzzella [35] stated that the achievable improvement in fuel efficiency depends strongly on the particular HEV system as well as on driving conditions. Control parameters that perform well under one set of conditions may lead to poor behavior under another. Hence, Sciaretta suggested adopting meaningful objective functions.

Evidently, this argument is also applicable to component design parameters. To conclude, HEV design parameters that are optimized for just one average-assumed drive cycle are not necessarily optimal for other drive cycles or the entire driving profile, which is the collectivity of all drive cycles in the lifetime of a vehicle. Single-cycle optimization might lead to so-called cycle beating: The optimized system design might be ideal for the given cycle, but may be suboptimal or not robust for other cycles. Further weaknesses of single-cycle optimizations include the neglect of deviations in driving patterns and life-time effects such as component degradation. On the other hand, the advantages are simplicity and lower requirements for computational resources.

Several methodologies to generate drive cycles have been published in the literature, most of them in the context of emission testing. Austin et al. [38] proposed to construct drive cycles by chaining categorized microtrips, which are defined as the driving activity between adjacent stops, including the leading period of idle. These microtrips are selected from observed data with the goal that the created cycle closely matches the characteristics of the data. Lin and Niemeier [39] criticize Austin's approach as lacking robustness and not reflecting of the stochastic nature of the

data. Thus, Lin and Niemeier suggested a mode-based cycle construction method, in which real world driving is viewed as a sequence of acceleration, deceleration, cruise, or idle modes. Markov process theory is used to describe the stochastic process. Tazelaar et al. [40] also created a method that generates drive cycles by resembling the characteristics of one original drive cycle in terms of frequency spectra and speed distribution.

The drive cycle generation tool (DCGT) proposed in this dissertation distinguishes itself from the above methods by operating entirely non-deterministically in terms of data. It does not use a database of recorded data or data snippets that are combined to form a cycle, rather all key parameters of a drive cycle are described stochastically with a probability density function. Drive cycles are created in a modular fashion by assigning stochastically-determined values to each key parameter. The modules are then assembled to form a drive cycle according to predetermined rules. Hence, the DCGT offers the following advantages that are essential for stochastic optimization:

- It is capable of producing an unlimited amount of drive cycles
- stochastic parameters can either be determined from observed data or be manually adjusted
- It generates driving profiles that represent stochastic nature of observed data
- It generates driving profiles that represent load profiles on the power train (duty cycle)

This section presents a new methodology to generate drive cycles for stochastic optimization. It is shown that DCGT generated drive cycles are comparable to recorded driving profiles in terms of frequency spectra, speed distribution, acceleration distribution and load characteristics of their corresponding duty cycles. Furthermore, the advantages of using the DCGT for stochastic optimization are investigated for a fuel cell powered HEV. Including life-time component effects, such as degradation and uncertainties of driving patterns, leads to improved and more comprehensive optimal solutions when compared to conventional single-cycle optimization methods. The proposed approach guarantees stable system operation throughout the simulation and for all scenarios of the driving profile.

Various terminologies are used in the literature in the context of this topic, as pointed out by Liaw and Dubarry [41]. Table 3.1 contains the definitions for terms used in this paper as well as alternative terminologies from the literature.

Table 3.1. Definition of the terminologies used in this study and alternative terminologies used in the literature.

Terminolgy	**Alternative Terminologies**	**Definition**
Drive Cycle	Driving Cycle	Set of data points representing vehicle speed versus time.
Duty Cycle	-	Set of data points representing power demand versus time.
Driving Pattern	-	Characteristics of a drive cycle influenced by internal and external factors such as the environment and driving behavior.
Driving Profile	Driving Cycle Profile	Collectivity of all drive cycles in the lifetime of a vehicle.
Driving Scenario	Driving Event	Subsection of a drive cycle distinguished by similar topographic environments, for example highway driving or urban driving.
Driving Pulse	Trip Snippet, Microtrip	Subsection of a drive cycle consisting of all data points between two idle periods.

3.1 Modular stochastic drive cycle generation

Driving profiles are dependent on a large set of properties such as topography, traffic, location, driving characteristics of the operator, and the environment. These dependencies generate immense complexity, which makes it virtually impossible to model each feature of a driving profile in a stochastic manner. Hence, some simplifying assumptions are required. The parameters presented in Table 3.2 were identified as the most crucial for describing the characteristics of a driving profile. Moreover, they are sufficient for the purpose of this study.

3.1.1 Drive Cycle

The DCGT methodology is based on a nested modularity concept, as illustrated in Fig. 3.1. The most outer module (module 1) is the DC itself. The drive cycle duration t_{DC} and its

Table 3.2. DCGT key parameters and their notations. DC: Drive Cycle, DS: Driving Scenario, VN: Velocity Noise.

Parameter	**Notation (fcn of)**	**Prob. Fcn (fcn of)**	**Unit**
Driving scenario	DS	$p_{DS_{occ}}(DS)$	$string$
DC duration	$t_{DC}(\{DS_n\})$	$p_{DC_t}(DS)$	s
DS duration	$t_{DS}(DS)$	$p_{DS_t}(DS)$	s
Acceleration	$a(DS)$	$p_a(DS)$	ms^{-2}
Deceleration	$d(DS)$	$p_d(DS)$	ms^{-2}
Cruising speed	$v_{CR}(DS)$	$p_v(DS)$	ms^{-1}
Cruising duration	$t_{CR}(DS)$	$p_{CR_t}(DS)$	s
Idle duration	$t_{ID}(DS)$	$p_{ID_t}(DS)$	s
Velocity noise	VN(DS)	$p_{VN_{occ}}(DS)$	$string$
VN amplitude	$A_n(DS)$	$p_{A_n}(DS)$	ms^{-1}
VN frequency	$\omega_n(DS)$	$p_{\omega_n}(DS)$	$(rad)s^{-1}$
VN phase	$\varphi_n(DS)$	$p_{\varphi_n}(DS)$	s

probability function p_{DC_t}(DS) are dependent on the DS categories (module 2) that are selected for the cycle. Overall, drive cycles consist of the following modules: driving scenarios (module 2), driving pulses (module 3) and pulse modules (module 4), which consist of acceleration, cruise, deceleration, idle, and velocity noise.

3.1.2 Driving Scenario

Driving scenarios are classified into five categories as proposed by [41]:

1. Stop-and-Go (SNG)
2. Urban (URB)
3. Suburban (SUB)
4. Rural (RUR)
5. Highway (HIG)

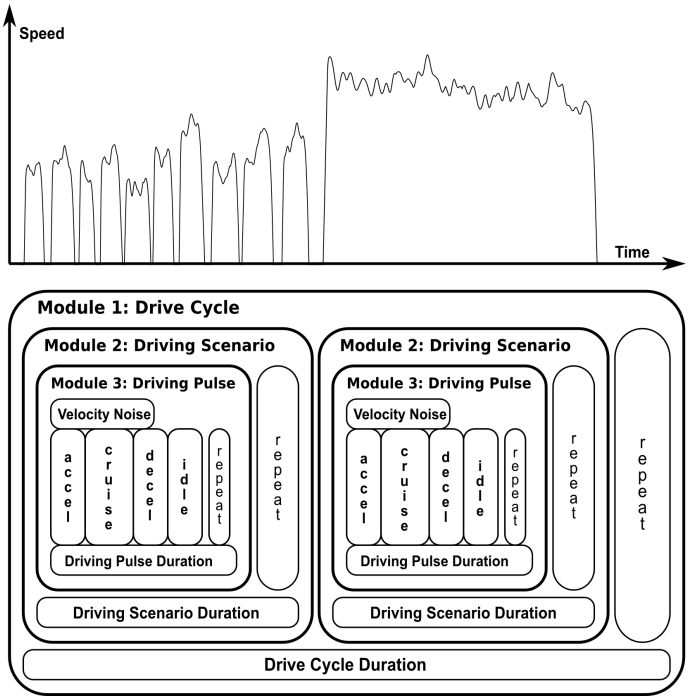


Figure 3.1. Graphical illustration of the 4-level nested modularity concept of the DCGT

The classification is intuitive. While URB, SUB, RUR and HIG are selected according to street topology, SNG can occur in any of these topologies during heavy traffic congestion. An occurrence probability $p_{DS_{occ}}(DS)$ is assigned to each driving scenario, with

$$\sum_{i=1}^{5} p_{DS_{occ}}(DS_i) = 100\% \tag{3.1.1}$$

The duration of one scenario is t_{DS} and the duration probability is p_{DS_t}(DS). The DCGT repeatedly generates driving scenarios until the following criteria is met:

$$\sum_{i=1}^{n} t_{DS}^{i} >= t_{DC} \tag{3.1.2}$$

3.1.3 Driving Pulse

A driving pulse (DP) consists of an initial acceleration phase, a cruising period, a deceleration phase, and an idle period as shown in Fig. 3.2. A typical driving pulse in an urban environment could be a trip from a traffic light to a stop sign, for example. The DP duration t_{DP} is determined using the following equation:

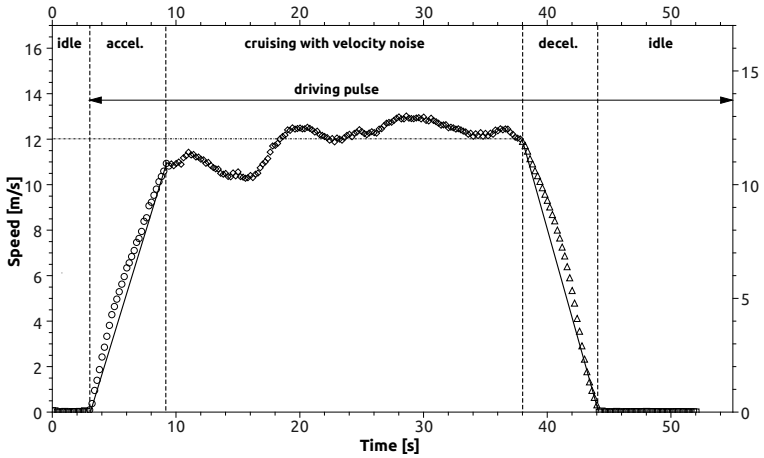


Figure 3.2. Partitioning of an urban driving pulse into four pulse modules (acceleration phase, cruising period and deceleration phase)

$$t_{DP} = \frac{v_{CR}^{t=0}}{a(DS)} + t_{CR}(DS) + \frac{v_{CR}^{t=t_{CR}(DS)}}{-d(DS)} + t_{ID}(DS) \tag{3.1.3}$$

$v_{CR}^{t=0}$ and $v_{CR}^{t=t_{CR}(DS)}$ are the vehicle speed at the beginning and at the end of the cruising phase, respectively. Typically, t_{DP} is very short for SNG traffic in a busy urban environment, and on the other extreme, very long for highway driving. The DCGT generates DPs until their accumulated duration is larger than the duration of the DS as defined by:

$$\sum_{i=1}^{n} t_{DP}^{i} >= t_{DS} \tag{3.1.4}$$

3.1.4 Pulse Modules

For simplification, acceleration and deceleration are assumed to be linear. An acceleration period ends when cruising speed is reached, and a deceleration period ends when the vehicle idles.

Velocity noise

As mentioned before, drive cycles are influenced by a large number of internal and external parameters. The vehicle speed during a cruising period fluctuates to a greater or lesser extent depending on these parameters. These speed fluctuations during a cruising period are referred to as *velocity noise*, and are viewed as a superposition of oscillations in this study. The advantage of this approach is that the velocity noise can be fully parameterized. It is then possible to obtain the

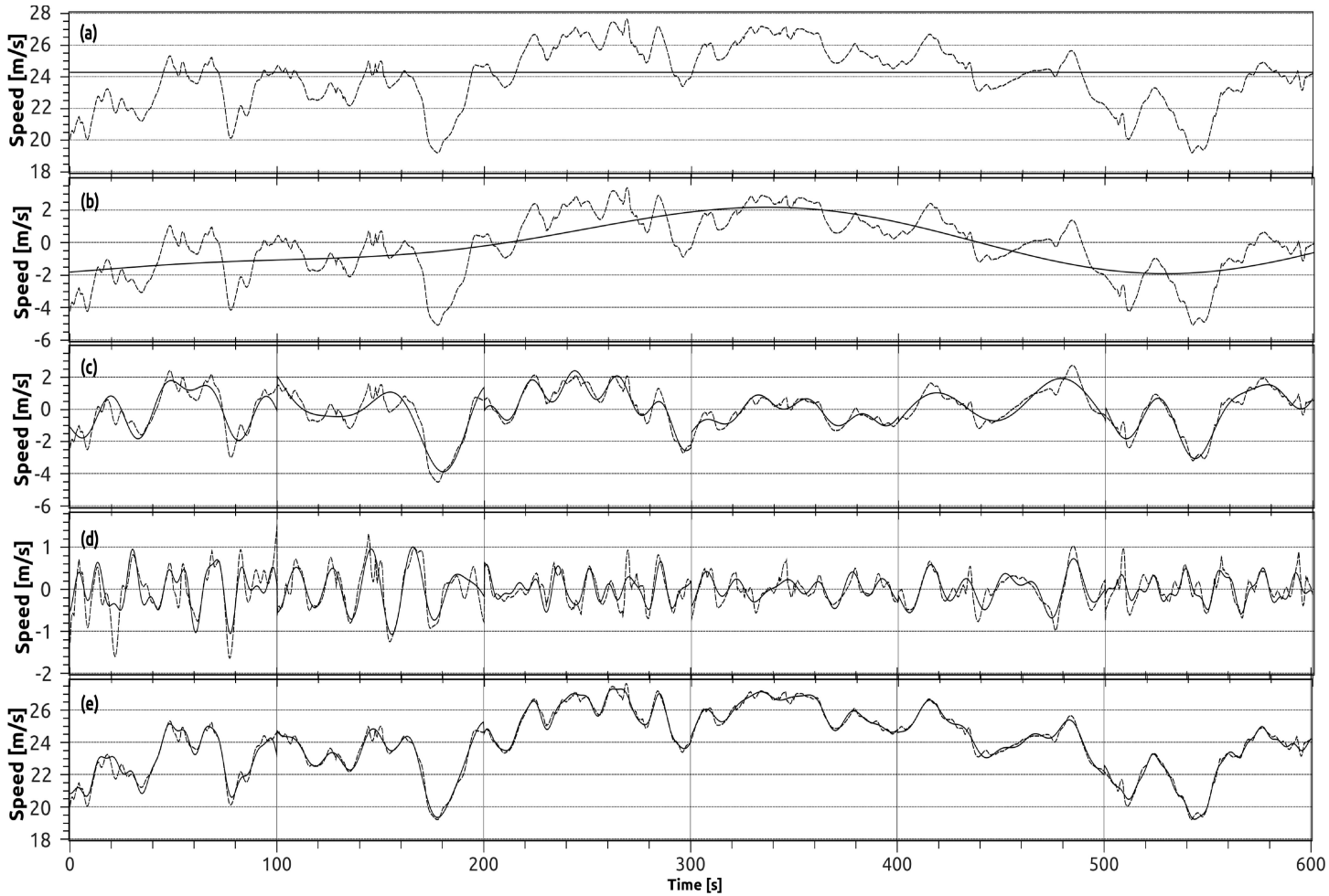


Figure 3.3. DCGT frequency decomposition methodology applied to driving data recorded on Interstate Highway H-1 in Hawaii, USA: (a) recorded data (dashed line) and average speed of recorded data (solid line), (b) velocity noise defined as speed over time minus average speed (dashed line) and its low frequency domain (solid line), (c) velocity noise minus low frequency domain (dashed line) and its medium frequency domain for 100s intervals (solid line), (d) velocity noise minus low and medium frequency domain (dashed line) and its high frequency domain for 100s intervals (dashed line), (e) recorded data (dashed line) and the sum of low, medium and high frequency domains plus average speed (solid line).

parameters and their probability functions from the data or adjust them manually. Velocity noise oscillations are categorized into three groups:

- **Low Frequency (LF) Noise (0-0.01Hz):**
  caused by terrain topology, traffic congestion, speed limits, construction areas, etc.
- **Medium Frequency (MF) Noise (0.01-0.25Hz):**
  caused by road topology, traffic flow, driving characteristics, etc.
- **High Frequency (HF) Noise (0.25-0.5Hz):**
  caused by road condition, lane switches, rapid driving maneuvers, spontaneous reactions, etc.

The oscillations for each frequency spectrum (FS) are modeled as a sum of three sinusoidal functions:

$$y_{FS}(t) = \sum_{i=1}^{3} (A_{FS}^{i} \cdot \sin(\omega_{FS}^{i} t + \varphi_{FS}^{i})) \tag{3.1.5}$$

Non-linear least square regression (NLLSR) is used to determine the amplitudes A_{FS}^{i}, the frequencies ω_{FS}^{i} and the phase φ_{FS}^{i} for each FS from recorded driving data [66]. The optimization problem for the regression can be stated as:

$$\min_{t} ||y_{FS}||_2^2 = \min_{t} (y_{FS,1}^2 + y_{FS,2}^2 + ... + y_{FS,m}^2) \tag{3.1.6}$$

with

$$\begin{aligned} y_{FS,1} &= f(t_1; A_{FS}^{i}, \omega_{FS}^{i}, \varphi_{FS}^{i}) \\ \vdots & \qquad \vdots \\ y_{FS,m} &= f(t_m; A_{FS}^{i}, \omega_{FS}^{i}, \varphi_{FS}^{i}) \end{aligned}$$

where m is the number of timesteps of a discrete DC. Initial guesses are picked for the unknown parameters and the following equation is defined:

$$\Delta\beta_j = y_{FS,j} - f(t_j; A_{FS}^{i}, \omega_{FS}^{i}, \varphi_{FS}^{i}) \tag{3.1.7}$$

Linear estimates for the parameter changes ΔA_{FS}^{i}, $\Delta\omega_{FS}^{i}$, and $\Delta\varphi_{FS}^{i}$ needed to reduce $\Delta\beta j$ to 0 are obtained using:

$$\Delta\beta_j = \left(\sum_{k=1}^{3} \left(\frac{\partial f}{\partial A_{FS}^k} \Delta A_{FS}^k + \frac{\partial f}{\partial \omega_{FS}^k} \omega_{FS}^k + \frac{\partial f}{\partial \varphi_{FS}^k} \varphi_{FS}^k \right) \right) \Bigg|_{t_k, (A_{FS}, \omega_{FS}, \varphi_{FS})} \tag{3.1.8}$$

for j=1,..,3. Eqn. 3.1.8 can be written in concise matrix form as

$$\Delta\beta = \mathbf{M}\Delta\lambda \tag{3.1.9}$$

where $\mathbf{M}$ is a $m \times 9$ matrix and $\lambda = \{A_{FS}^i, \omega_{FS}^i, \varphi_{FS}^i\}$. Eqn. 3.1.9 can now be solved for the offset $\Delta\lambda$. $\lambda + \Delta\lambda$ results to a new $\Delta\beta$. The process is iteratively repeated until a convergence criteria is met.

The regression is executed for all driving scenarios as illustrated in Fig. 3.3, in which the approach is performed for highway driving data recorded on Interstate Highway H-1 in Hawaii, USA. The first step is to isolate the velocity noise from the cruising period by subtracting the average speed (Fig. 3.3(a)). Next, the low frequency noise is identified using NLLSR and subsequently separated from the velocity noise (Fig. 3.3(b)). For the following steps, the remainder of the velocity noise is partitioned into segments of equal duration (100s). The regression parameters of the medium frequency noise are then determined for each segment and the medium frequency noise is filtered from the remainder of the velocity noise (Fig. 3.3(c)). The same procedure is executed once again to identify and filter the high frequency noise (Fig. 3.3(d)). Fig. 3.3(e) shows the original data set versus the velocity noise modeled with Eqn. 3.1.5 using the obtained parameters. The latter closely resembles the orginal curve.

3.1.5 Probability Functions

A probability function (or probability density function) is assigned to each parameter that is used to describe the driving profile. Most probability functions are two-dimensional; however, they can also be multi-dimensional if they are dependent on multiple parameters. Probability functions are derived by interpolating data from the driving profile analysis. The probability function is then adjusted for the use of a uniform random number generator for the range [0,1]. Fig. 3.4 illustrates this approach for the *acceleration* parameter in an urban environment. Data of 100 acceleration periods, which was collected during urban driving in Honolulu, Hawaii, is categorized into an arbitrary number of bins. 10 bins are used in this example. NLLSR is used to fit the data to the following equation, assuming that the data has Gaussian characteristics:

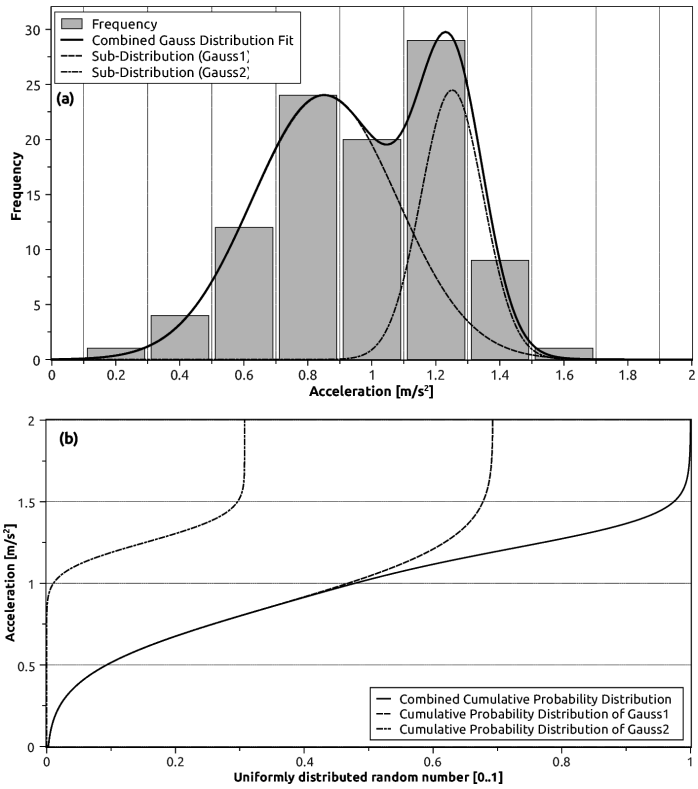


Figure 3.4. (a) Summation of Gaussian distributions fitted to a histogram of 100 urban accelerations, (b) cumulative probability distribution derived from the Gaussian distribution and adapted for a uniform random number generator

$$f(x;\alpha_n,\sigma_n,\mu_n)=\sum_{i=1}^{n}\left[\frac{\alpha_i}{\sigma_i\sqrt{2\pi}}e^{\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)}\right] \tag{3.1.10}$$

where α is a scaling factor, σ^2 is the variance, μ is the mean and n is the number of overlapping Gaussian distributions. $n=2$ is chosen in Fig. 3.4(a). Next, the cumulative distribution function $\Phi(x)$, which describes the probability of a random variable in $(-\infty,x]$, is determined using:

$$\Phi(x)=\int_{-\infty}^{x}\left(\sum_{i=1}^{n}\left[\frac{\alpha_i}{\sigma_i\sqrt{2\pi}}e^{\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)}\right]\right)dx \tag{3.1.11}$$

$$=\frac{1}{2}\sum_{i=1}^{n}\left[\alpha_i\left[1+erf\left(\frac{x-\mu_i}{\sigma_i\sqrt{2}}\right)\right]\right] \tag{3.1.12}$$

where erf is the Gauss error function, a special function of sigmoidal shape. Now, $\Phi(x)$ needs to be normalized to fit the desired probability range.

$$\Phi^*(x) = \frac{\Phi(x)}{\lim\limits_{x\to\infty}(\Phi(x))}) \quad , \quad \Phi^*(x) \in [0,1] \tag{3.1.13}$$

Fig. 3.4(b) shows the cumulative probability distribution $P(x)$ of the $acceleration$ data adjusted for a random number generator. The displayed function is derived as the inverse of Eqn. 3.1.13 and is solved numerically.

$$P(\gamma) = {\Phi^*}^{-1}(\gamma) \tag{3.1.14}$$

3.2 Tool Validation

10 hours (489km) of driving data was acquired in the City and County of Honolulu in March and April of 2011. The test vehicle used for this study is a standard compact car (1200kg, 92hp @ 5500rpm, 5-speed-manual) with an internal combustion engine. The collected driving data incorporates all possible driving scenarios (9% SNG, 38% URB, 17% SUB, 11% RUR and 25% HIG). A Qstarz^{TM}BT-Q1000eX GPS device was used for data acquisition. The device is based on a MTK II GPS chipset featuring a tracking sensitivity of -165dBM, 66 Channel tracking and 5Hz data logging. The manufacturer lists a velocity accuracy of $\pm$ 0.1m/s. A Savitzky-Golay smoothing filter is used to eliminate the high frequency noise ($>$0.5Hz) caused by inaccuracies of the GPS device and by short interruptions of the satellite connection when driving underneath bridges [67].

The probability functions of all parameters in Table 3.2 were extrapolated from the data according to the methodology described in Section 3.1 and implemented into the DCGT. The tool is now capable of creating an unlimited amount of drive cycles based on the stochastics of this driving profile. Fig. 3.5 shows recorded and generated DPs for different driving scenarios. Evidently, the generated cycles possess characteristics similar to the original data for all five scenarios in terms of acceleration speed, pulse duration, average speed, velocity frequencies, and others.

A more in-depth investigation of the comparability of the recorded data and the created driving profile can be achieved by looking at their speed-acceleration frequency distributions (SAFDs). SAFD analysis provides the needed information about the time proportions of a driving profile [42]. Fig. 3.6 shows the SAFD for both the recorded data and 10h of data created by the DCGT. Two frequency peaks at $(x = 0\frac{m}{s^2}, y = 23\frac{m}{s})$ and $(x = 0\frac{m}{s^2}, y = 17\frac{m}{s})$ are characteristic for the SAFD in Fig. 3.6(a). Similar peaks at $(x = 0\frac{m}{s^2}, y = 22\frac{m}{s})$ and $(x = 0\frac{m}{s^2}, y = 17\frac{m}{s})$, and overall a very similar frequency distribution, are noticeable in the SAFD plots in Fig. 3.6(b). The recorded driving data features an abrupt frequency drop at speeds higher than 29m/s. Considering

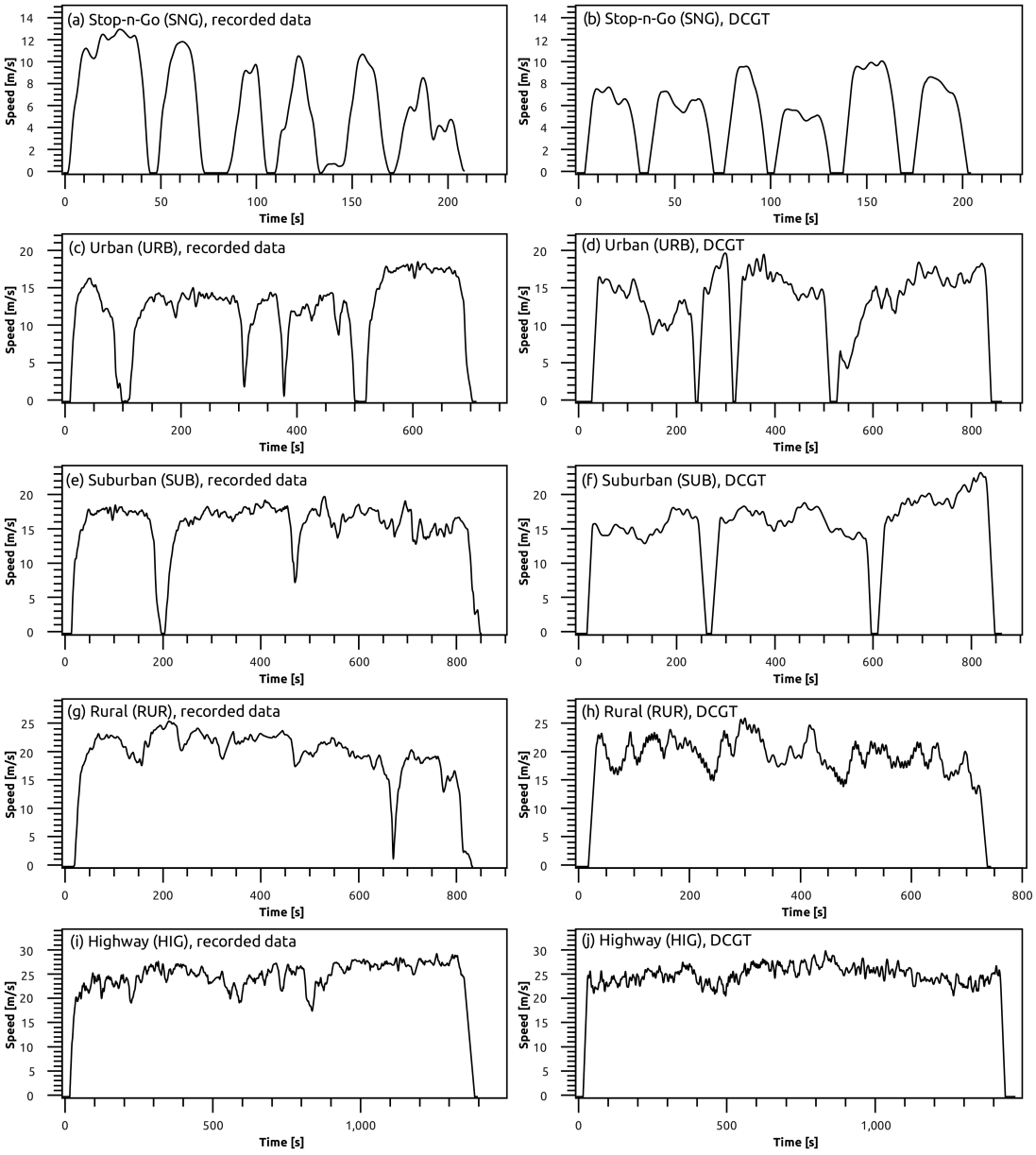
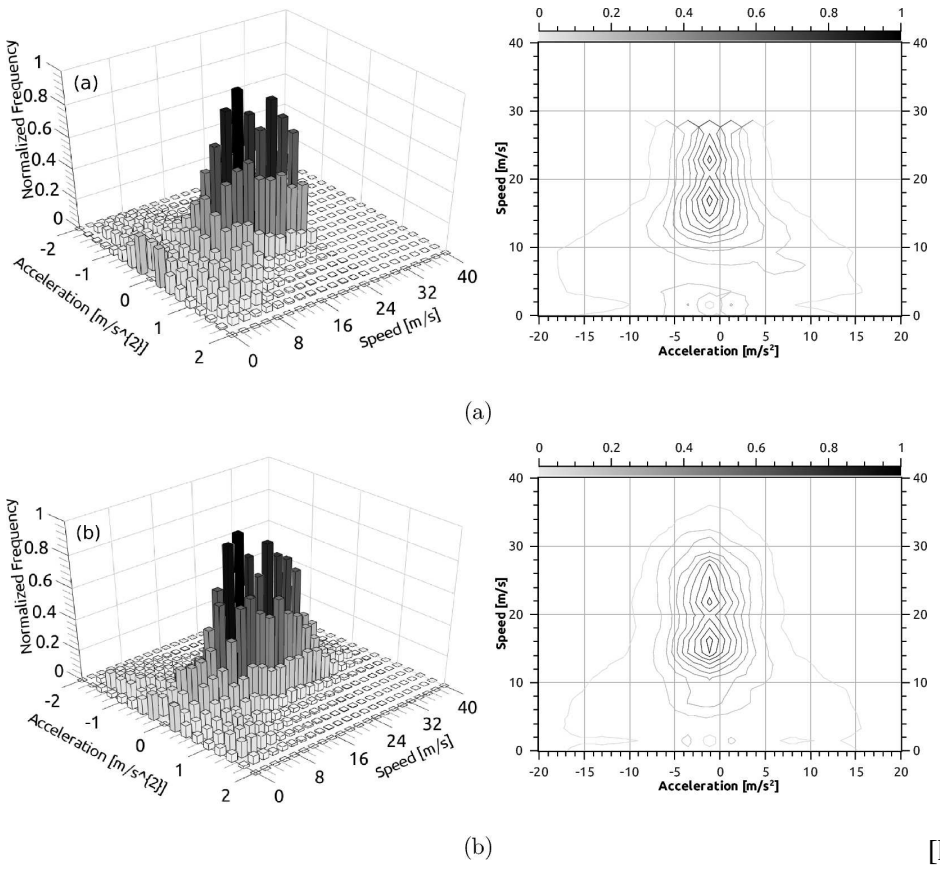


Figure 3.5. Recorded (left column) and DCGT generated (right column) driving pulses for 5 the different driving scenarios

that the DCGT probability functions are based on Gaussian distributions, it is not possible to model such a rapid probability drop-off without implementing constraints. However, the DCGT SAFD is fairly broad and low in the discussed region and therefore not expected to have any considerable effects on the optimization results of the vehicle.

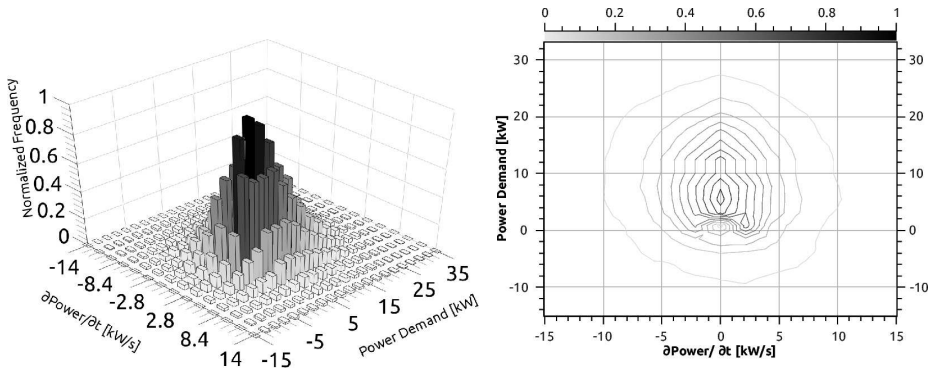


[h]

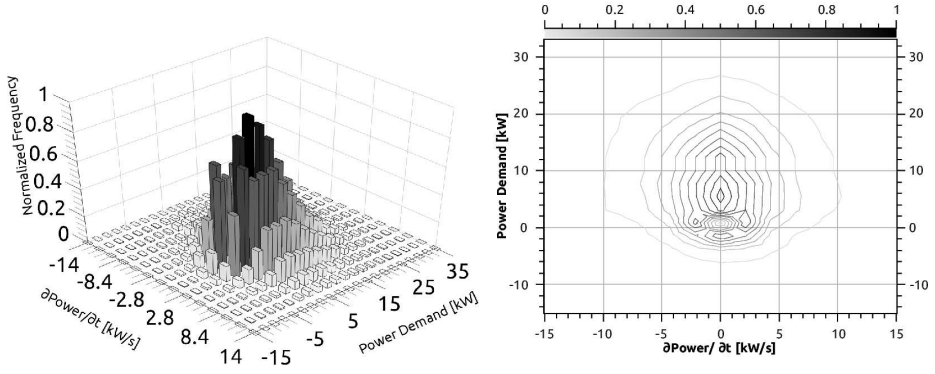
Figure 3.6. A speed-acceleration frequency distribution (SAFD) histogram and contour plot for 10h of recorded drive cycle data vs. 10h of DCGT generated drive cycle data. The data excludes the zero-acceleration/zero-speed bin to prevent visual distortion due to its dominance. (a) 10h of real drive cycle data recorded in the City and County of Honolulu, USA, (b) 10h of DCGT generated drive cycle data based on a stochastic driving profile extrapolated from recorded data in (a).

The physical impact of a driving profile on a vehicle can be examined by visualizing the frequency domain of the power demand. The power demand over time is obtained by converting a drive cycle into a duty cycle by considering the dynamic properties of the vehicle. The vehicle dynamics model (Section 2.1) calculates the incremental change in position of the vehicle and the external forces for each iteration of the simulation. A driving profile can now be converted into an estimated duty profile.

Fig. 3.7 visualizes the frequency spectra of both converted driving profiles in a power - delta power frequency distribution (PdPFD) plot. Again, similar distributions of both, the recorded profile and the DCGT profile, are observed. Both distributions feature one prominent power frequency peak at $(x = 0\frac{kW}{s}, y = 7kW)$. Additional common characteristic frequency peaks are



(a)



(b)

Figure 3.7. A power - delta power frequency distribution (PdPFD) histogram and contour plot for a duty cycle profile generated from 10h of recorded drive cycle data vs. 10h of DCGT generated drive cycle data. The data excludes the zero-power/zero-delta-power bin to prevent visual distortion due to its dominance. (a) Estimated power frequency distribution of 10h of real drive cycle data recorded in the City and County of Honolulu, USA, (b) Estimated power frequency distribution of 10h of DCGT generated drive cycle data based on a stochastic driving profile.

located at $(x = 0\frac{kW}{s}, y = 2kW)$, $(x = 2\frac{kW}{s}, y = 3kW)$ and $(x = -2\frac{kW}{s}, y = 3kW)$. The latter two are characteristic of the acceleration and deceleration habits of the driver. The frequency distribution in the x-y plane of Fig. 3.7(a) possesses a slightly wider foundation than its counterpart in Fig. 3.7(b). This discrepancy is most likely caused by inaccuracies of the parameter fitting methodology for the probability functions. However, the stochastic impact during a vehicle optimization is expected to be minor due to its comparatively small frequencies in this regions.

Chapter 4

Large-Scale Optimization on GPUs

4.1 Optimization in Engineering Design

Optimization is the act of obtaining the best result under given circumstances. In mathematical terms, optimization is the process of finding conditions that give the maximum or minimum value of a function [1]. Design optimization of complex systems requires immense data processing and a tremendous number of calculations [2]. In the advent of high-speed computers, optimization methodologies are becoming increasingly popular in engineering design [3]. Engineering optimization is prevalent in various fields, such as structural design, shape optimization, topological optimization, and logistics. Nevertheless, optimization of dynamic engineering systems is exceptionally expensive in terms of computation speed and is therefore still constrained by related limitations.

If mathematical optimization techniques are used for engineering design, the optimization problem can usually be stated in the following form:

$$
\begin{aligned}
& \min_{x} f(x) && && (4.1.1)\\
\text{subject to} \quad & g_j(x) \geq 0, && j = 1, 2, ..., J\\
& h_k(x) = 0, && k = 1, 2, ..., K\\
& x_i^{(L)} \leq x_i \leq x_i^{(U)}, && i = 1, 2, ..., N\\
& x \in \mathbb{S}
\end{aligned}
$$

The function to be minimized or maximized is the objective function $f(x)$. The design variables are represented by the column vector $x = (x_1, x_2, ..., x_N)^T$ [3]. $g_j(x) \geq 0$ and $h_k(x) = 0$

define the two types of constraints, the first of which are inequality constraints and the second of which are equality constraints. Design variables can be real-valued ($x \in \mathbb{R}^n$), integer ($x \in \mathbb{Z}^n$), binary ($x \in \{0,1\}^n$) or categorical (e.g., $x \in \{\text{blue, red, white, green}\}^n$) [4]. Categorial spaces can usually be replaced by integer spaces by assigning integers to each category. $\mathbb{S}$ is the search space or solution space, which is the set of all possible solutions. Again, $\mathbb{S}$ can be real-valued, integer, binary, categorical, or a combination of all, since its composition depends on the properties of the design variables. For example, assuming that the design optimization of a structural beam consists of two design variables $x = (x_1 = \text{type of cross section}, x_2 = \text{web thickness})$, x_1 would be element of a categorical or integer search space and x_2 of a real-valued search space. The overall search space would be a cartesian product of both solution sets: $\mathbb{S}_x \in (\mathbb{S}_{x_1} \times \mathbb{S}_{x_2})$ with $x_1 \in \mathbb{Z}$ and $x_2 \in \mathbb{R}$.

Design variables can be classified into two categories: *independent design variables* and *dependent design variables*. Independent design variables are all crucial design parameters represented by the design vector x. Dependent design variable are functions of independent design variables and are therefore not required to be included in the design vector. Constraints are conditions that a solution to an optimization problem must satisfy. The restrictions that must be fulfilled to produce an acceptable design are called *design constraints*. Constraints that are physical limitations on the state of the system are termed *functional constraints* [1]. For example, the capacity of a battery pack would be restricted by design constraints, whereas functional constraints would be used to limit the SOC range. Since functional constraints are constraints on the state of the system, they can be classified into three categories:

1. *Threshold constraints*: If a state variable s surpasses its threshold constraint $g_t(s)$, it is set equal to the threshold constraint.

   Require: $s \leq g_t(s)$
   if $s > g_t(s)$ **then**
   $s = g_t(s)$
   end if

2. *Elimination constraints*: If a state variable s surpasses its elimination constraint $g_e(s)$, the fitness evaluation is stopped and the candidate is eliminated from the set of candidates.

   Require: $s \leq g_e(s)$
   if $s > g_e(s)$ **then**
   kill candidate

end if

3. *Penalty constraints*: If a state variable s surpasses its penalty constraint $g_p(s)$, a penalty p(s) is imposed on the objective.

   Require: $s \leq g_p(s)$
   if $s > g_p(s)$ **then**
   $f(x) \leftarrow f(x) + p(s)$
   end if

Engineering design problems are frequently characterized by multiple, often times conflicting, objectives. For example, reducing the weight of an airplane wing while minimizing the cost of materials are two conflicting objectives. Light materials, such a composite carbon fiber, are usually more costly than their cheaper, but heavier alternatives. There are two common approaches to solve multi-objective optimization problems. First, all objectives can be combined into a single aggregate objective function using a weighting function. Secondly, a multi-dimensional candidate solution (Pareto optimal solution) can be determined followed by a decision maker selecting an ideal candidate according to his preferences. The advantage of the latter approach is that no a priori information or weighting function is needed for the actual optimization [5].

4.2 FCHEV Optimization Problem

The FCHEV optimization problem, which is predominantly used in this dissertation, is stated as:

$$\min_{p} \sum_{i=1}^{k} m^{i}_{f_{H_2}}(p) \tag{4.2.1}$$

$$p = [P_{mt}, n_{fc}, Q] \tag{4.2.2}$$

The objective in Eqn. 4.2.1 is the overall hydrogen consumption in the lifetime of the vehicle. The stopping point parameter k is the number of performed drive cycles until the lifetime mileage of the vehicle d_{lt} is reached. k is determined by algorithm 4.1.

The selected independent design parameters are the rated motor power p_1=P_{mt}, the number of fuel cells in the stack p_2=v_{st} and the capacity of the battery pack p_3=Q. Hence, the design vector p contains the crucial sizing parameters for the power sources and power suppliers of the FCHEV power train. The choice of design parameters allows for the study of the ideal power train

Algorithm 4.1 Stopping point parameter k

$d = 0, k = 1$
while $d \leq d_{lt}$ **do**
  $d = d + d_{DC_k}$
  $k = k + 1$
end while

composition for individual (application and driver-specific) driving profiles, for specialized driving profiles (pure city driving, pure highway driving, etc.), and for universal driving profiles (all-purpose vehicle). It is then possible to quantify the efficiency gain form using an optimized power train for certain driving profiles versus universal or non-ideal power trains. Furthermore, degradation and uncertainties of the component models can be investigated.

From an engineering point of view, it often makes sense to discretize search spaces for design parameters of physical systems to save computation time. This simplification is feasible since it is not the mathematically optimal solution that is sought after, but the solution that is optimal while still being practical. For example, fuel cells in a fuel cell stack are usually identical in terms of their size specifications. It is physically not viable to integrate fuel cells of different sizes into one stack. Therefore, it is not required to describe the design parameter v_{st} as a real-valued variable. The search space and the computation time are drastically reduced if v_{st} is described as an integer.

The following equations are an example of how the design constraints of the proposed optimization problem can be reasonably described:

$$0hp < P_{mt} \leq 200hp, \qquad P_{mt} \in \mathbb{Z}^{+}_{200} \tag{4.2.3}$$

$$0 < n_{fc} \leq 600, \qquad n_{fc} \in \mathbb{Z}^{+}_{600} \tag{4.2.4}$$

$$0Ah < Q \leq 15Ah, \qquad (Q \times 100) \in \mathbb{Z}^{+}_{1500} \tag{4.2.5}$$

All three design variables are discretized into partitions of equal length. The minimum discretization length is chosen according to technical and economical feasibility. In the presented example, the cardinality of the overall search space can be determined as follows:

$$\mathbb{S} = \quad \mathbb{S}_{P_{mt}} \times \mathbb{S}_{n_{fc}} \times \mathbb{S}_{Q} \tag{4.2.6}$$

$$\Rightarrow \quad |\mathbb{S}| = \quad |\mathbb{S}_{P_{mt}}| \times |\mathbb{S}_{n_{fc}}| \times |\mathbb{S}_{Q}| = 200 \times 600 \times 1500 = 1.8 \times 10^{8} \tag{4.2.7}$$

The cardinality is equivalent to the number of possible solutions for the optimization problem.

Alongside design constraints, meaningful functional constraints have to be selected for most dynamic systems. Table 4.1 lists the functional constraints implemented in the FCHEV model. The vehicle speed cannot be zero due to Eqn. 2.1.7 and Eqn. 2.1.8. To remedy this, a threshold

Table 4.1. Functional constraints of the FCHEV model

State variable	**Type of constraint**	**Constraint condition**	**Consequence**
Vehicle speed v	Threshold constraint	$v \leq 1 \times 10^{-8}$	$v = 1 \times 10^{-8}$
Battery state-of-charge SOC	Elimination constraint	$20\% \leq SOC \leq 80\%$	kill candidate
Battery degradation $\frac{Q^*}{Q}$	Elimination constraint	$\frac{Q^*}{Q} \leq 80\%$	kill candidate
Fuel cell voltage v_{fc}	Elimination constraint	$v_{fc} < 0$	kill candidate
Power demand P_{total}	Penalty constraint	$P_{total} > P_{mt}$	$m_{f_{H_2}} = m_{f_{H_2}} \cdot p(P_{total})$

constraint is used to replace the stopping speed with a value that is very close to zero ($10^{-8}\frac{m}{s}$). The battery state-of-charge is constrained within its safe range of operation [6]. An elimination constraint is applied since SOCs outside of the defined range indicate that the battery is too small for the intended task. Battery damage has to be avoided. The capacity fade of the battery pack is limited to 20% according to DOE guidelines [7]. At higher levels, the degradation rate increases rapidly, and the power train may reach a critical state. An elimination constraint is selected to prevent this scenario. Numerical instabilities cause negative fuel cell voltages, which indicate that the proposed fuel cell stack is sized too small. Since these instabilities are highly non-linear, it is very challenging to design a meaningful penalty function. To simplify matters, an elimination constraint is used instead. Finally, a penalty constraint is used for the overall power demand of the vehicle. If the power demand is higher than the maximum rated power of the motor, the requested power cannot be supplied. The effect might be negligible if it only occurs a few times in the lifetime of a vehicle. However, if the effect is a repeatedly occurring phenomenon, the proposed design would be inadequate. In this case, a penalty function can account for the phenomenon according to the number of occurrences.

4.3 Optimization with Genetic Algorithms

Genetic algorithms are selected as the optimization method of choice in this dissertation for the following reasons:

- GAs are meta-heuristic optimization routines (Section 1.1) and are therefore parallelizable on the GPU architecture.

- GAs are efficient and effective at finding optimal solutions in huge search spaces.
- GAs are mostly generic and can be applied to many types of problems. Little knowledge and information is needed about the problem domain [8].
- The efficiency of GAs can be significantly improved by adjusting the optimization parameters towards the characteristics of the optimization problem.
- GAs have been proven to be effective optimization tools for a large number of applications [9].

The GAs proposed in this study consist of the operators shown in Fig. 4.1, which are population initialization, selection, crossover, mutation, and termination. The following paragraphs give an

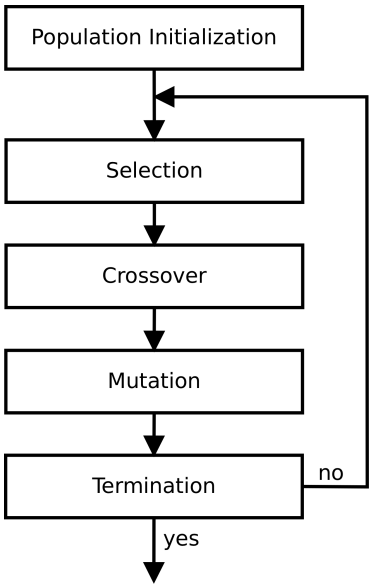


Figure 4.1. Operators of the genetic algorithm

overview of the operator's specifications used in this study.

4.3.1 Population Initialization

The ideal population size depends on the nature of the optimization problem and the hardware architecture [10, 11]. Section 5.2.1 and 5.2.2 describe how the optimal population size can be determined. Depending on the convergence properties of the optimization problem, the ideal population size can range from as little as 20 members to more than 100,000 members [12].

Initial population members can be either seeded in regions where optimal solutions are expected, equally distributed over the search space, or randomly generated. In this study, the latter option is chosen. A uniform random number generator ([0,1]) is used to create the initial population, as seen in Algorithm 4.2.

Algorithm 4.2 Random initialization of GA population members

Require: $g_1 \leq x_n \leq g_2, 0 \leq r \leq 1$

for $i = 1 \rightarrow n$ **do**

$r = rand()$

$x_i = g_1 \cdot r \cdot (g_2 - g_1)$

end for

4.3.2 Selection

Selection is the process of choosing the fittest individual candidates of a population for later crossover (breeding). The fitter a candidate, the more times it is likely to be selected. The selection process in this study is implemented as follows:

1. Fitness Evaluation
2. Sigma Scaling
3. Sorting - Odd-even transposition sort
4. Elitism
5. Fitness proportionate selection

Fitness Evaluation

Each candidate must be evaluated in the selection process. Therefore, the FCHEV model is executed for each population member. The results (objective) are saved and used in the following operator steps.

Sigma Scaling

Sigma scaling keeps the selection pressure (the degree to which highly fit candidates are allowed many offspring) relatively constant during the optimization process [13]. The scaled fitness

$f_i^*(x)$ is a function of its fitness before scaling $f_i(x)$, the population's mean fitness $\bar{f}$, the standard deviation of the population's fitness σ, and a constant scaling coefficient s.

$$f_i^*(x) = 1 + \frac{f_i(x) - \bar{f}}{s \cdot \sigma} \qquad , \sigma > 0 \tag{4.3.1}$$

When the standard deviation is high, the probability of being selected is very similar for each candidate. On the other hand, fit candidates have a higher probability of being selected when the standard deviation is low. This usually happens during the later part of the optimization, when the population is more converged. The selection pressure is increased and evolution can continue.

Odd-Even Transposition Sort

Algorithm 4.3 Odd-even transposition sort

Require: n is even

```
for i = 0 → n do
  for j = 0 → n/2 − 1 do
    if f[2j] > f[2j + 1] then
      swap(f[2j], f[2j + 1])
    end if
  end for
  for j = 0 → n/2 − 1 do
    if f[2j + 1] > f[2j + 2] then
      swap(f[2j + 1], f[2j + 2])
    end if
  end for
end for
```

Odd-even transposition sort (OETS) is used to sort the candidates according to their fitness and boasts several advantages, which are:

- OETS is parallelizable on the GPU architecture.
- OETS is simple to implement.
- OETS is applicable to different array sizes if even-numbered.

The algorithm performs compare-exchange operations, alternating between odd-even pairs and even-odd pairs, until convergence is reached. The worst case performance of the algorithm is O(n^2). Algorithm 4.3 shows a pseudocode implementation of odd-even transposition sort.

Elitism

In order to not lose the fittest candidates, elitism reserves spots for the best parent candidates in the population of the next generation [14]. The elite parent candidates are transfered without any modifications, i.e. crossover and mutation. Elitism prevents a population from moving away from an already-found optima. Unless otherwise specified, the number of elite parents is 4 in this study.

Fitness Proportionate Selection

Fitness proportionate selection (FPS) is analogous to a roulette wheel with each section of the wheel proportional in size to the fitness of the candidate. Hence, fitness proportionate selection is also known as roulette-wheel selection. The functioning of the operator is described in Algorithm 4.4. First, a cumulative probability is assigned to each candidate of the population. The cumulative probability range is scaled to [0,1]. Secondly, candidate pairs are determined to generate offspring for the next generation using a random number generator ([0,1]). Candidate pairs are created for each design parameter of the design vector. Hence, the total number of candidate pairs is the total population size minus the number of elite parents times the number of design parameters.

4.3.3 Crossover

Crossover is an operator that creates an offspring from two selected parent candidates. There are several crossover techniques, most of which are binary crossover techniques, such as one-point crossover, cut-and-slice, and uniform crossover. Linear crossover is used as the crossover operator in this study, since it is an efficient, real-valued technique with easy implementation. The design parameters of the offspring are the mean values of the equivalent design parameters of the parents:

$$x_i^{new} = \frac{x^*_{i_{male}} + x^*_{i_{female}}}{2} \tag{4.3.2}$$

Algorithm 4.4 Fitness proportionate selection

$f_{sum} = 0$
for $i = 0 \rightarrow n$ **do**
  $f_{sum} = fsum + f_i(x)$
end for
$p_f[0] = f_0(x)/f_{sum}$
for $i = 1 \rightarrow n$ **do**
  $p_f[i] = p_f[i-1] + f_i(x)/f_{sum}$
end for
Number of elite parents: $n_{EP} = 4$
for $i = 0 \rightarrow (2n - n_{EP}) \cdot |x|$ **do**
  $r_1 = rand(), r_2 = rand()$
  for $j = 1 \rightarrow n$ **do**
    if $p_f[j-1] \leq r_1 < p_f[j]$ **then**
      $c_1 = j$
    end if
    if $p_f[j-1] \leq r_2 < p_f[j]$ **then**
      $c_2 = j$
    end if
  end for
  $x^*_{i_{male}} \leftarrow x_{c_1}, x^*_{i_{female}} \leftarrow x_{c_2}$
end for

4.3.4 Mutation

Mutation is an operator that modifies the values of some design parameters from their initial states. Mutation is used to ensure candidate diversity and prevent the population from converging to a local optimum. Inspired by biological mutation, the occurrence of mutation is restricted to a mutation probability. If the mutation probability is set too high, the genetic algorithm will perform similarly to a random search. *Gaussian mutation* is used in this study to alter the floating point design parameters of selected offspring. A Gaussian distributed value is generated and added to the selected design parameters. If the resulting value falls outside the constraints, the mutation procedure will be repeated. Generating Gaussian-distributed numbers is not a simple task on GPUs, however, uniform random number generation is supported in the CUDA environment. The *Box-*

Muller transform creates pseudo-Gaussian-distributed numbers by mapping a uniform distribution to a standard normal distribution [15]. Two random, uniformly distributed variables r_{u_1} and r_{u_2} in the interval [0,1] are transformed to become normal distributed variables (r_{n_1}, r_{n_2}) using:

$$r_{n_1} = \sqrt{-2\ln r_{u_1}} \cdot \cos(2\pi r_{u_2}) \quad (4.3.3)$$

$$r_{n_2} = \sqrt{-2\ln r_{u_1}} \cdot \sin(2\pi r_{u_2}) \quad (4.3.4)$$

Algorithm 4.5 Floating point Gaussian mutation

```
for i = 0 → (2n − n_EP) do
  r_g = rand()
  if r_g < CandidateMutationRate then
    for j = 0 → |x| do
      r_c = rand()
      if r_c < ParameterMutationRate then
        m = f(boxmuller(), x_j)
        x_j ← x_j + m
      end if
    end for
  end if
end for
```

Algorithm 4.5 explains the implementation of the mutation operator. Two mutation rates are defined: First, the *candidate mutation rate (CMR)* is the probability that mutation occurs on one or more design parameters of a population candidate. Second, the *parameter mutation rate (PMR)* is the probability that mutation occurs on a design parameter if the candidate it belongs to has been selected for mutation. If a design parameter is selected for mutation, the operator adds a Gaussian-distributed value to the parameter. The added value is the result of the Box-Muller transform times a parameter-specific scaling function.

4.3.5 Termination

GAs are either terminated when a certain convergence criteria is met, or after a pre-defined number of iterations. If a priori knowledge about the design problem is limited, it might be difficult to define reasonable convergence criteria. To prevent infinite looping, a maximum number of

generations should be defined. Infinite looping might occur if the population gets stuck in a local minimum.

4.3.6 Convergence Testing

Typical symptoms of underperfoming optimization procedures include the algorithm getting trapped in local minima and slow convergence rates in areas with extremely low gradients. Both symptoms often emerge as conflicting objectives when addressed during the tuning of meta-heuristic optimization algorithms. On the one hand, it is desirable to maintain a wide-spread population over the entire search space to locate all possible minima. On the other hand, a large concentration of solution candidates within close proximity of a minima increases the convergence rate of determining the minima's exact location. Hence, an effective optimization algorithm is capable of escaping from local minima while featuring a sufficient convergence rate.

The characteristics of optimization search spaces of real-world applications are often unknown. As a result, the optimization routine of choice needs to be preliminarily assessed for extreme test cases to obtain a feeling for both its convergence capabilities and convergence speeds. Convergence capabilities of optimization procedures are typically evaluated by using literature benchmarking functions [16]. In the following paragraphs, the convergence performance of the proposed GA is tested for various functions of different complexity, dimensionality, and optimization resistance. The selected functions feature extreme search spaces with either a large number of local minima, extensive areas of low gradient, or a combination of both. It is shown that the proposed GA performs remarkably well for all test scenarios. In all of the presented tests, the algorithm managed to determine the global minima within the required accuracy while exhibiting a reasonable convergence rate at all times.

Rosenbrock Function

The Rosenbrock function, also known as the second De Jong function, is a non-convex function commonly used for performance testing of meta-heuristic optimization algorithms [17].

$$f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right] \qquad \forall x \in \mathbb{R}^n \tag{4.3.5}$$

Fig. 4.2 illustrates the characteristics of the Rosenbrock function for n=2. The global minimum is located within a wide-spread valley of low gradients. The challenge for the algorithm is to quickly converge to the global minimum despite the nearly flat contour of the valley.

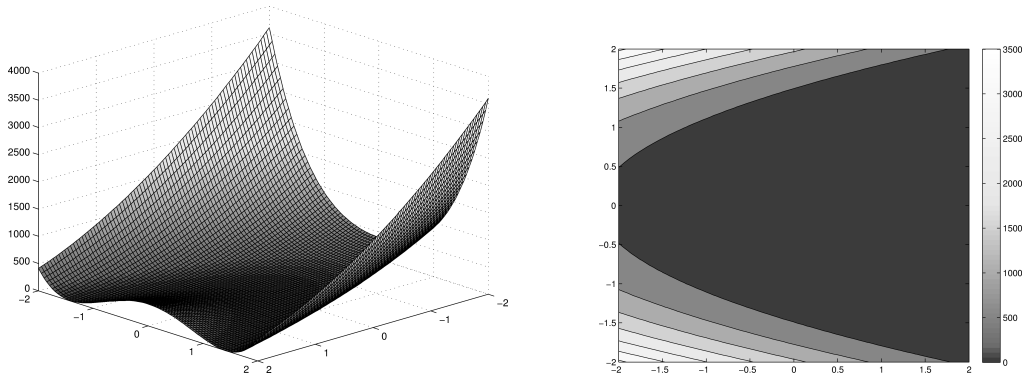


Figure 4.2. Rosenbrock's function for n=2, -2.048 $\leq x_i \leq$ 2.048. The global minimum is located at $f(x_i = 1) = 0$.

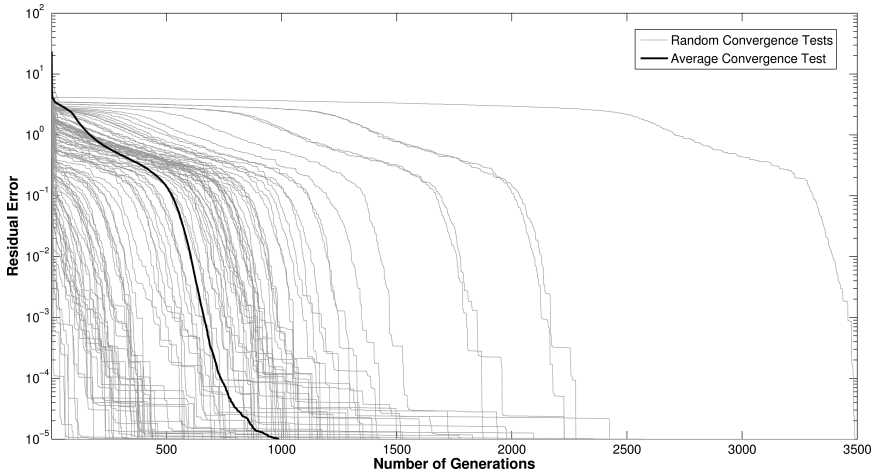


Figure 4.3. 100 GA convergence tests and average convergence process for Eqn. 4.3.5 with n=3, -2.048 $\leq x_i \leq$ 2.048 and random, uniformly distributed initial populations. Population size=200, CMR=0.9, PMR=0.4.

Fig. 4.3 shows the convergence curves of 100 performed optimization runs with random initial populations and the resulting average convergence curve. It is noticeable that all 100 runs converge to an accuracy of less than 10^{-5} needing between 60 and 3500 generations to accomplish the task. Furthermore, a bend in the convergence curve is predominant, which can be explained by the properties of the Rosenberg function. Finding the valley and converging to a function value close to the global minimum can be quickly achieved by the GA. On on the other hand, determining the exact location of the minima is more challenging and therefore causes the observed decrease in the convergence rate.

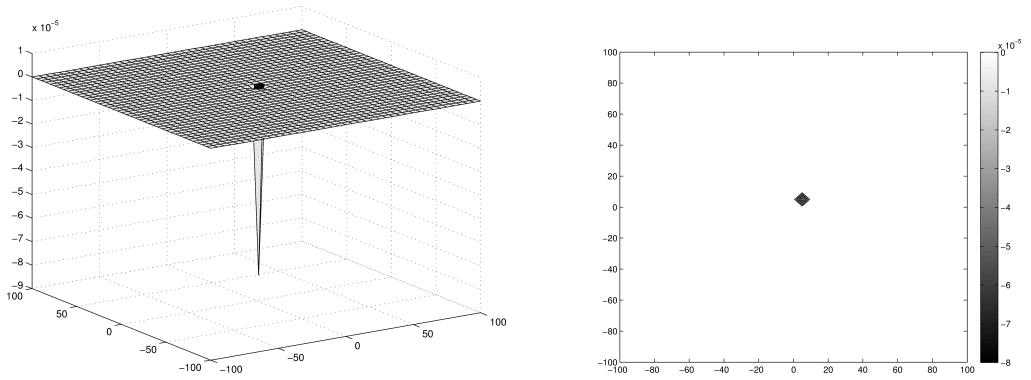


Figure 4.4. Easom function for -100 $\leq x_i \leq$ 100. The global minimum is located at $f(\pi, pi) = -1$.

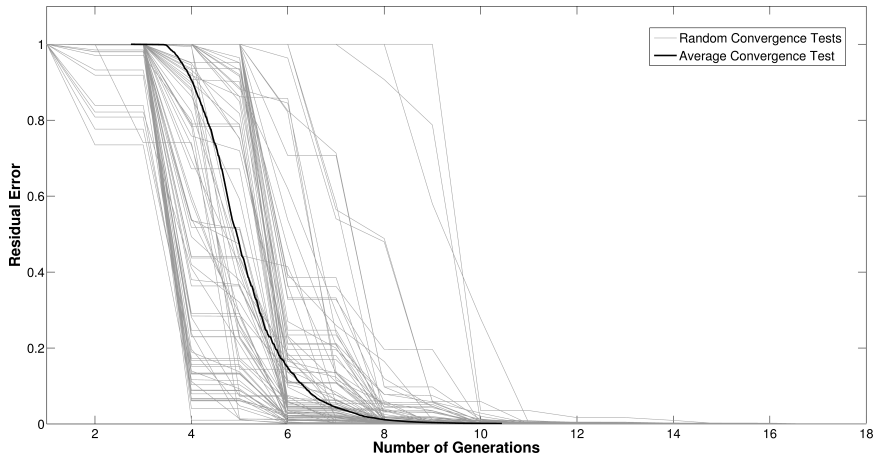


Figure 4.5. 100 GA convergence tests and average convergence process for Eqn. 4.3.6 -100 $\leq x_i \leq$ 100 and random, uniformly distributed initial populations. Population size=200, CMR=0.9, PMR=0.4.

Easom Function

The Easom function (Eqn. 4.3.6) is a 2-dimensional benchmarking function for which the majority of the search space is equivalent to a nearly flat plane with no considerable gradient. Its only minima is located at $f(\pi, \pi) = -1$.

$$f(x_1, x_2) = -cos(x_1)cos(x_2)exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2) \tag{4.3.6}$$

Fig. 4.5 illustrates the GA performance when tested with the Easom function. All of the 100 performed tests converged very quickly within less than 15 generations. It can be concluded that the mutation operator of the GA prevents the algorithm from stagnation when the crossover operator is non-functional due to equivalent function values of all population members.

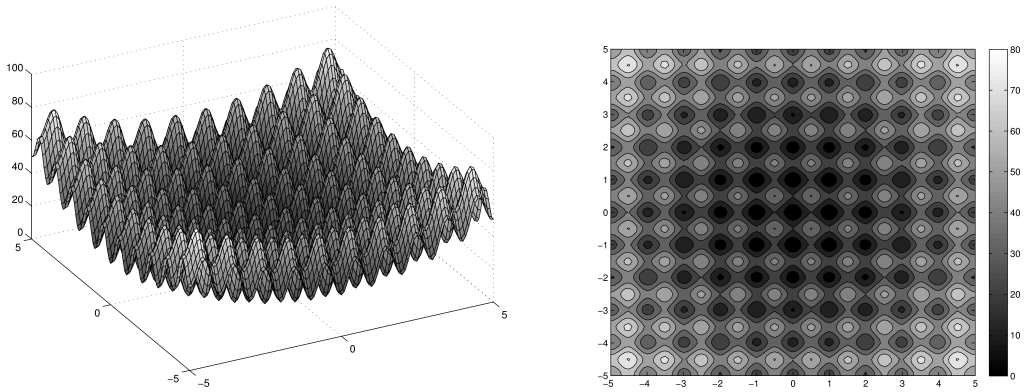


Figure 4.6. Rastrigin's function for n=2, -5.12 $\leq x_i \leq$ 5.12. For A=10, the global minimum is located at $f(x_i = 0) = 0$.

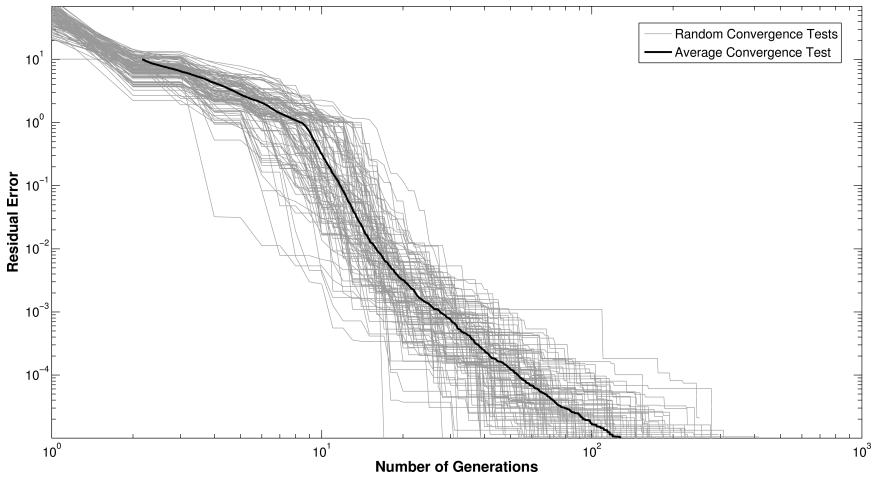


Figure 4.7. 100 GA convergence tests and average convergence process for Eqn. 4.3.7 with n=3, -5.12 $\leq x_i \leq$ 5.12 and random, uniformly distributed initial populations. Population size=200, CMR=0.9, PMR=0.4.

Rastrigin Function

The Rastrigin function (Eqn. 4.3.7) is a non-linear, multimodal function commonly used for benchmarking. It is characterized by a large number of deep local minima with similar function values, as illustrated in Fig. 4.6. The Rastrigin function is considered a very difficult optimization problem due to its large search space and a high probability of getting entrapped in a local minima.

$$f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10cos(2\pi x_i)\right] \qquad \forall x \in \mathbb{R}^n \tag{4.3.7}$$

Fig. 4.7 shows the convergence curves for a 3-dimensional search space. All test functions converge between 14 and 420 generations. The convergence curves approach the global minimum in a stepwise manner, which is caused by the algorithm constantly discovering local minima through mutation until the global minimum is found.

Langermann Function

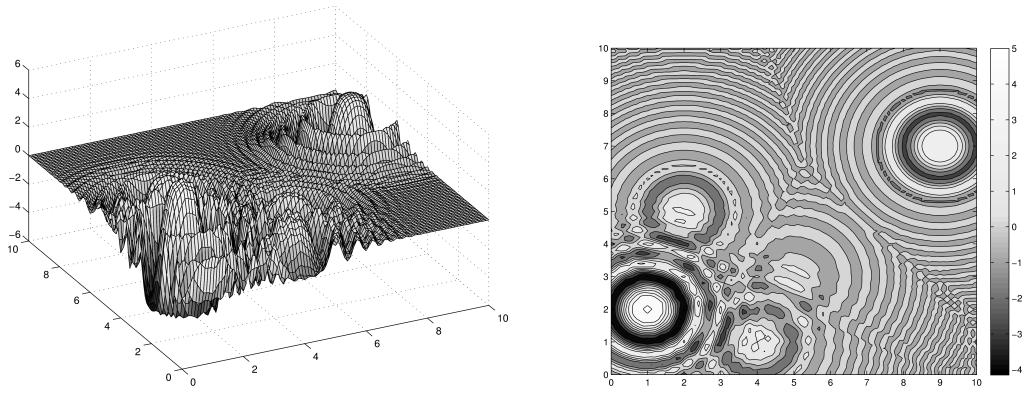


Figure 4.8. Langermann function for n=2, m=5, $a_{ij} = [3, 5, 2, 1, 7; 5, 2, 1, 4, 9]$, $0 \leq x_i \leq 10$.

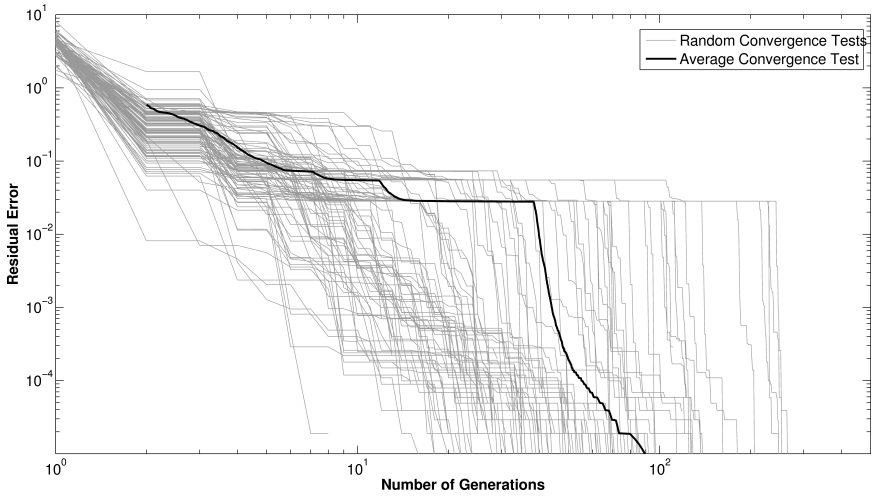


Figure 4.9. 100 GA convergence tests and average convergence process for Eqn. 4.3.8 with n=3, $0 \leq x_i \leq 10$ and random, uniformly distributed initial populations. Population size=200, CMR=0.9, PMR=0.4, $a_{ij} = [3, 5, 2, 1, 7; 5, 2, 1, 4, 9; 1, 2, 5, 2, 3]$ and global minimum at $f(2.793, 1.597, 5.307) = -4.156$.

The Langermann function (Eqn. 4.3.8) is a multimodal benchmarking function that features both flat sections with very small gradients and areas with a multitude of nested, ring-shaped minimas (Fig. 4.8). It therefore demands universal applicability from the optimization algorithm.

$$f(x) = \sum_{i=1}^{m} c_i \exp[-\frac{1}{\pi}\sum_{j=1}^{n}(x_j - a_{ij})^2]\cos[\pi\sum_{j=1}^{n}(x_j - a_{ij})^2] \quad \forall(x,c) \in \mathbb{R}^n, a \in \mathbb{R}^{n\times m} \quad (4.3.8)$$

Fig. 4.9 shows the results of 100 performance tests using the Langermann function. Again, all optimization runs converge within 9 to 270 generations. The average convergence curve features a relatively long period with a zero convergence rate before approaching the global minimum, since the ring-shaped characteristics of the local minima hamper the effectiveness of the crossover operator. However, the GA manages to determine the optimal solution at all times.

4.4 FCHEV Optimization Methodology

The following requirements are needed to be fulfilled by the FCHEV optimization methodology:

- **Objective:** Determine the best system design in terms of minimizing the overall fuel consumption for a given driving profile.
- **Simulation length:** Simulate the entire vehicle lifetime for each candidate (100,000km).
- **Evaluation data:** Use driving profiles to account for effects caused by variations in driving patterns.
- **Lifetime effects:** Account for battery degradation during lifetime simulation.
- **Optimization time:** Achieve reasonable computation time until convergence criteria is met ($\lesssim$ 2 days).
- **Energy management:** Optimize energy management for each drive cycle simulation to achieve comparability of all candidates.

A two-level optimization methodology is proposed to determine the ideal system configuration while achieving the design goals. Fig 4.10 illustrates the framework of the optimization method. The methodology consists of two genetic algorithm optimization routines executed in two

loops, an outer loop and an inner loop. The outer loop finds the optimal component design parameters, which are defined by the design vector in Eqn. 4.2.2. The objective is to minimize the fuel consumption for the entire lifetime of the vehicle (Eqn. 4.2.1). There are several ways to achieve this in terms of data partitioning.

1. All driving data from the lifetime of the vehicle is created as one large drive cycle up front and fed to the simulation. The model then simulates the total lifetime of the vehicle without interruptions.

2. The DCGT is integrated with the vehicle simulation and generates the drive cycle data on-the-go. Hereby, the DCGT only creates the necessary data for one timestep at a time on the fly. The simulation is stopped when the overall mileage reaches the lifetime mileage.

3. The lifetime driving data is split into partitions of equal array length, equal mileage, or similar topology (e.g. driving scenarios). A new data partition is created each time the simulation has finished evaluating the previous data partition. The final system state of the previous simulation is saved and used as the new initial state for the next simulation. This process is repeated until the overall mileage is greater than or equal to the anticipated lifetime mileage.

Option 1 requires a lot of memory space for the large data array. Modern GPUs currently do not provide enough fast memory to efficiently handle arrays of that size. Option 2 is very efficient in terms of memory utilization. However, due to the lack of a data history, it is very challenging to design an EMS that ensure comparability between candidates. Option 3, on the other hand, is very suitable for an efficient CUDA implementation and a simple EMS integration. The maximum array size of the data partition can be determined to fully utilize the fast memory space of the GPU and thereby minimize the memory latency between host and device. In the proposed optimization procedure, the DCGT generates drive cycles that consist of only one type of driving scenario. This uniformity eases the task of finding the ideal energy management for the drive cycle.

The inner optimization loop determines the ideal EMC configuration for each drive cycle simulation of each candidate. Since the inner optimization is nested into the outer optimization loop, the computation effort increases exponentially. However, the nested implementation is necessary in order to guarantee comparability of the population members in the outer loop. Assuming the EMC parameters are not optimized for each drive cycle, then low-potential candidates with efficient EM parameters might outperform high-potential candidates with inefficient EM parameters. High-potential candidates might appear as low-potential candidates and will have an unproportionally low

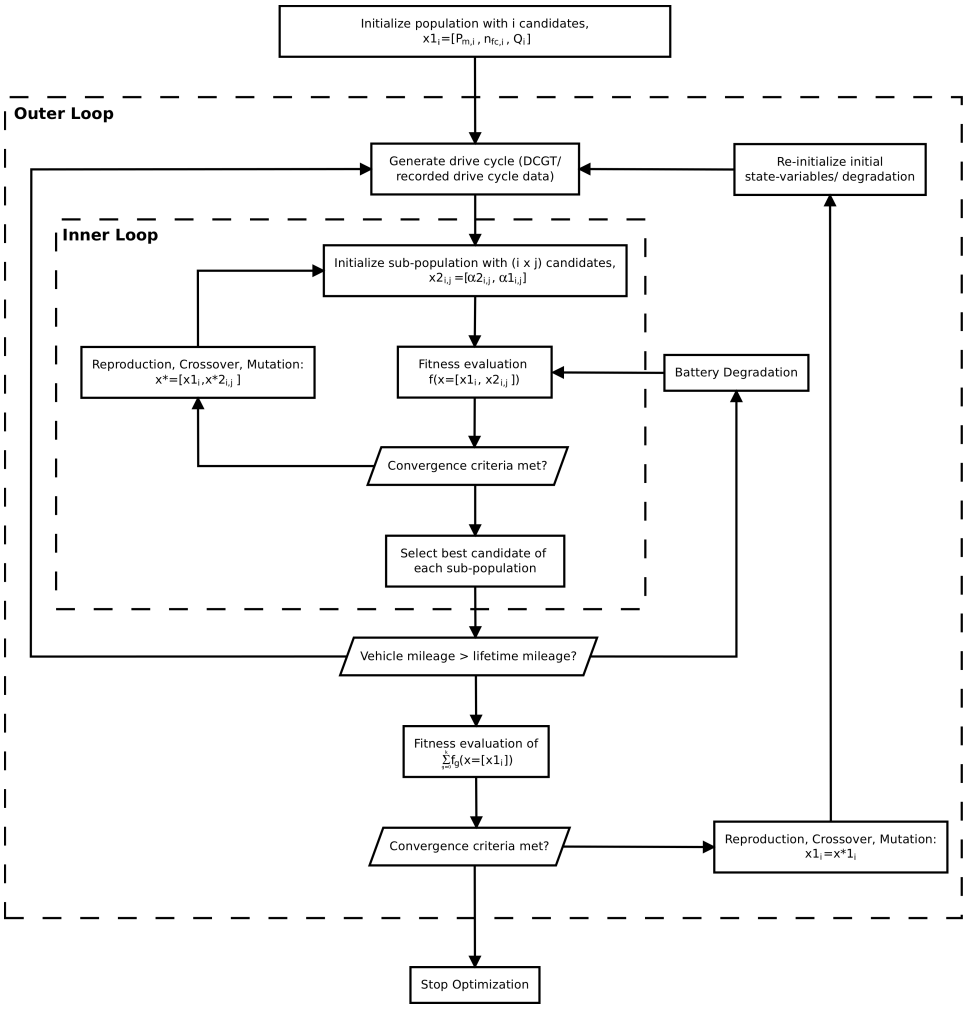


Figure 4.10. Two-level optimization framework using two genetic algorithms: The outer loop finds the ideal component design configuration p for 100,000km of drive cycle data; the inner loop determines an ideal energy management setup for each drive cycle simulated for each population member.

reproduction probability. Capable genes might therefore not be represented in the next generation. Furthermore, high-potential design candidates, which have performed well in previous generations, might perform worse in the following generations because of suboptimal EMC parameters. Hence, the entire GA routine might not converge due to inconsistencies in the fitness evaluation. As a result, the inner EMC optimization loop is absolutely essential.

To use a genetic algorithm for the inner loop, it is necessary to create sub-populations for each candidate of the outer optimization loop. The overall population size of the optimization

problem thereby increases to:

$$n_{total} = n_{outer} \cdot n_{inner} \tag{4.4.1}$$

The objective of the inner loop optmimization is stated as:

$$\min_{q} m_{f_{H_2}}(q) \tag{4.4.2}$$

$$q = [\alpha_1, \alpha_2] \tag{4.4.3}$$

where $m_{f_{H_2}}$ is the hydrogen consumption for one drive cycle, and q is the design vector containing the EMC parameters α_1 and α_2, which are used in Eqn. 2.5.1. Reasonable constraints for the design vector x can be formulated as:

$$0 \leq \alpha_1 \leq 0.5, \qquad \alpha_1 \in \mathbb{Z}^+_{500} \tag{4.4.4}$$

$$0.5 \leq \alpha_2 \leq 1.5, \qquad \alpha_2 \in \mathbb{Z}^+_{1000} \tag{4.4.5}$$

The best candidate of an inner loop optimization is the one that has the lowest fuel consumption while maintaining a well-balanced battery SOC. If the SOC of the final state is smaller than the SOC of the initial state then the battery acted as an active power source and not just as a buffer. The extra energy provided by the battery must be accounted for in the objective function in order to ensure comparability of the candidates. The fuel consumptions of the best candidates are summed until the lifetime mileage is reached. During this process, the model keeps track of the battery utilization, which is needed to determine the battery's state of degradation. For each generated drive cycle, the degradation proceeds individually for each candidate. The state of degradation is set back to zero for each new iteration of the outer loop. All other state variables are also re-initialized. The optimization is stopped when the convergence criteria is met or if the maximum number of generations is reached.

4.4.1 Cardinality of the Search Space

The search space cardinality $\mathbb{S}_{total}$ of the proposed two-level optimization routine can be estimated by the following equation:

$$|\mathbb{S}_{total}| = \underbrace{\left(|\mathbb{S}_{P_{mt}}| \times |\mathbb{S}_{n_{fc}}| \times |\mathbb{S}_{Q}|\right)}_{\text{Cardinality of outer loop search space}} + \underbrace{\left[(|\mathbb{S}_{\alpha_1}| \times |\mathbb{S}_{\alpha_2}|) \cdot i\right]}_{\text{Cardinality of inner loop search space}} \cdot k \cdot n_{convergence} \tag{4.4.6}$$

where i is the number of population members of the outer loop, which is equivalent to the number of optimizations performed for each drive cycle in the inner loop. k is number of drive cycles in

the lifetime of the vehicle, and $n_{convergence}$ is the number of generations of the outer loop until the convergence criteria is met. The cardinality of the outer loop search space is given in Eqn. 4.2.7. Eqn. 4.4.5 and 4.4.5 define the search spaces of the design parameters of the inner loop. n_{outer} typically lies in the interval $\{20..800\}$. The mileage of a DCGT generated drive cycle is usually between 5km and 100km. If the overall lifetime of the vehicle is set to 100,000km, then k lies in the interval $\{1000..20000\}$. The minimum number of generations of the outer loop is 2, and the maximum number is limited to 100. Hence, the interval range for $n_{convergence}$ is $\{2..200\}$. The overall cardinality of the total search space is therefore estimated to be an element of the following interval:

$$|\mathbb{S}_{total}| \in \quad \{\big(1.8 \times 10^8 + (500 \cdot 1000 \cdot 20) \cdot 1000 \cdot 2\big) \,.. \quad (4.4.7)$$
$$\big(1.8 \times 10^8 + (500 \cdot 1000 \cdot 800) \cdot 20000 \cdot 100\big)\}$$
$$\rightarrow |\mathbb{S}_{total}| \in \quad \{2 \times 10^{10} \;\; .. \;\; 8 \times 10^{14}\} \quad (4.4.8)$$

4.5 GPU Implementation

In order to optimally implement the methodology proposed in Section 4.4, it is crucial to understand the CUDA programming model. A CUDA kernel is executed by a large array of

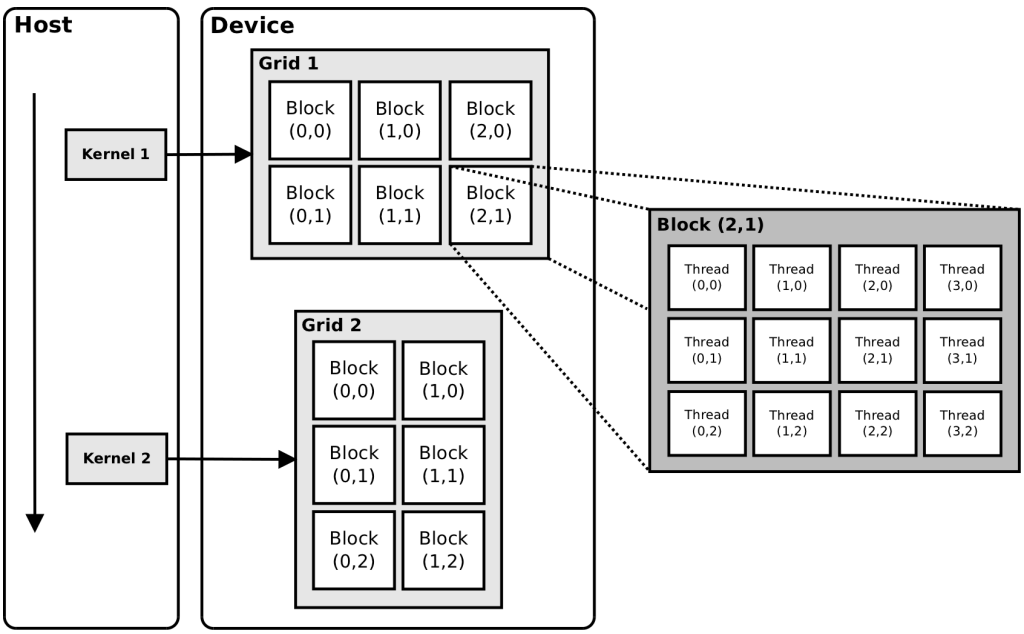


Figure 4.11. The CUDA programming model consisting of grids, blocks and threads.

threads. All threads run the same program, but with different input data (SIMD). Threads are usually organized into a two-level hierarchy, as shown in Fig. 4.11 [18]. When a kernel is *invoked*, it is executed on the GPU by a *grid*. A grid consists of one or more *blocks*, which again contain the *threads*. All blocks have the same number of threads assigned to them. A thread block is a batch of threads that allows for sharing data between the threads through shared memory. Thread blocks also enable synchronizing the execution of the threads within the block. However, threads from different blocks cannot interact; they operate independently.

Each multiprocessor of the GPU can only execute a limited number of threads at the same time, therefore, thread blocks are divided into *warps*. A warp is a group of threads within a block that are executed together. Each thread block is mapped to one or more warps. If the number of threads in a block is not a multiple of the warp size of the GPU, then the computational occupancy of the GPU is not fully utilized. Hence, an efficient CUDA implementation needs to be designed according to the structure of the CUDA programming model.

Two different GPU architectures have been used in this dissertation, which are single-GPUs and multi-GPUs. Single-GPUs are typically consumer-grade graphic cards or specially designed parallel processing units, which are embedded in a workstation computer. A multi-GPU cluster is a computer cluster in which each node is equipped with a GPU. The implementation of the proposed optimization routine slightly differs according to the architecture. The fitness evaluations of the sub-populations are the most computationally demanding tasks because of their enormous multitude. Hence, it is highly desirable to parallelize this evaluation process. Since the fitness evaluation process is extremely scalable, it can not only be executed in parallel on the multiprocessors of a GPU, but it can also be split between numerous devices.

Fig. 4.12 illustrates the implementation for a single-GPU architecture. First, the population for p is initialized on the host. Sub-populations for q are then initialized for each candidate of the main population. Each sub-population is assigned to a separate thread block on the device with the number of threads in a block equaling to the number of candidates in the sub-population. All blocks are assigned to a single grid. The FCHEV model and the GA of the inner optimization loop are implemented within the kernel, which is executed by the grid. Drive cycles are generated by the DCGT on the host and transfered to the global memory of the device. New grids are then initialized on the device for every simulated drive cycle. After each iteration of the outer loop, the fitness level (overall fuel consumption) of each block is saved to the host memory. Next, the outer loop GA optimization is executed on the host. A new population is created, split into sub-populations and send to the device. This process is repeated until convergence is achieved.

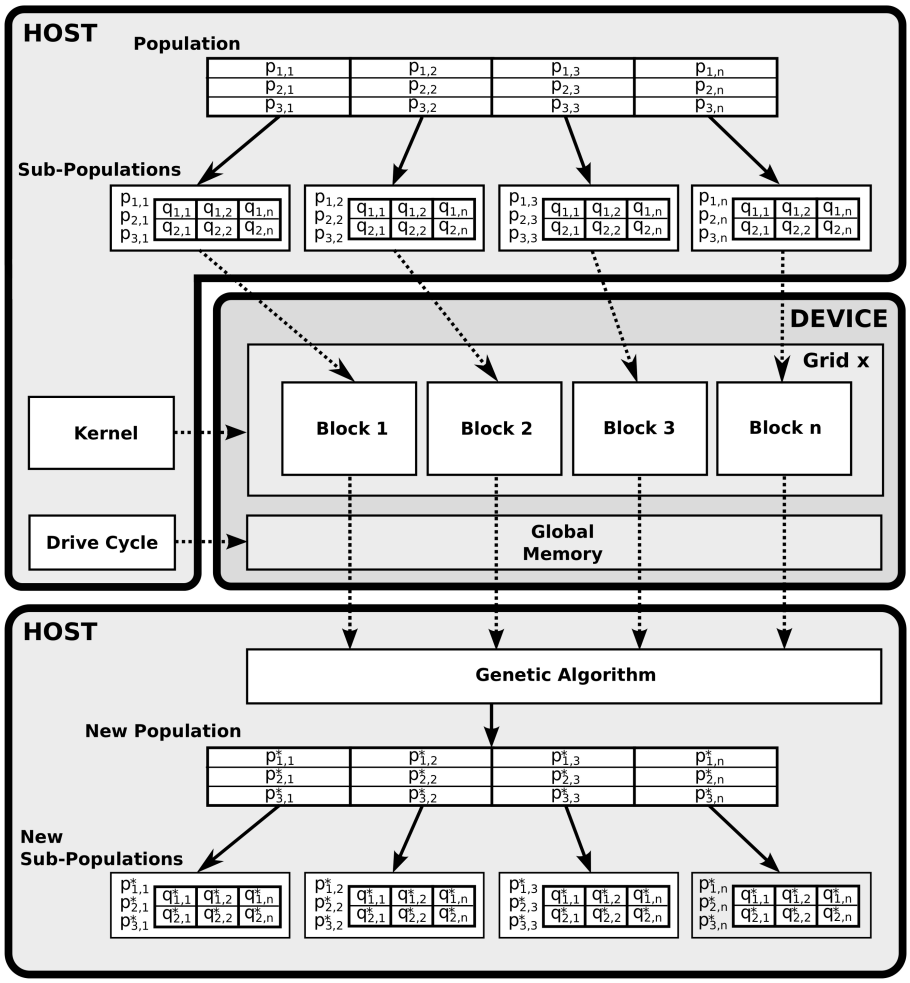


Figure 4.12. 2-level optimization methodology implemented on a single-device GPU. Each sub-optimization is performed in separate blocks, which enables scalability.

Fig. 4.13 shows the implementation for a multi-GPU architecture. The population for p is initialized on the host. Sub-populations are then initialized for each candidate of the main population. Next, the sub-populations are evenly distributed into groups equivalent to the number of available devices. Each group is represented by one grid on each device. The sub-populations of a group are assigned to blocks in the corresponding grid, and the same kernel and the same drive cycles are executed on all devices in parallel. The results of each block are copied to the host memory after each iteration of the outer loop. The outer loop GA optimization is executed on the host. A new population is created, split into sub-populations, which are grouped according to the number of available devices, and sent to the device. This process is repeated until convergence is achieved.

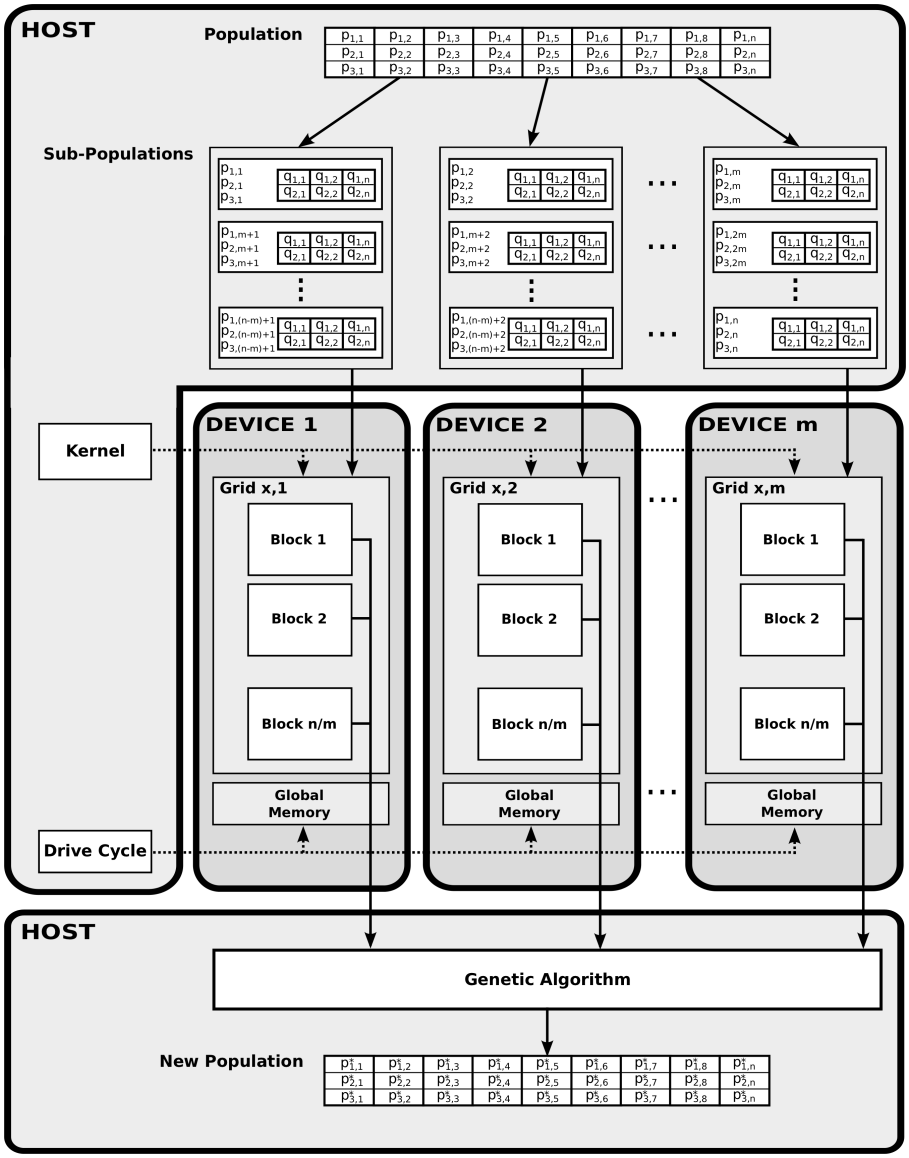


Figure 4.13. 2-level optimization methodology implemented on a multi-GPU architecture. Each sub-optimization is performed in separate blocks. The blocks are evenly split into grids, which are executed on multiple devices in parallel.

Grid synchronization is essential for the multi-GPU implementation to ensure convergence of the GAs.

The synchronization of threads within one block is illustrated in Fig. 4.14. The kernel, which is simultaneously launched by each thread, consists of two parts: The first part is the FCHEV simulation and the second part is the inner loop GA optimization. The evaluation of the FCHEV is an *embarrassingly parallel* workload, which means that no dependencies exist between the parallel

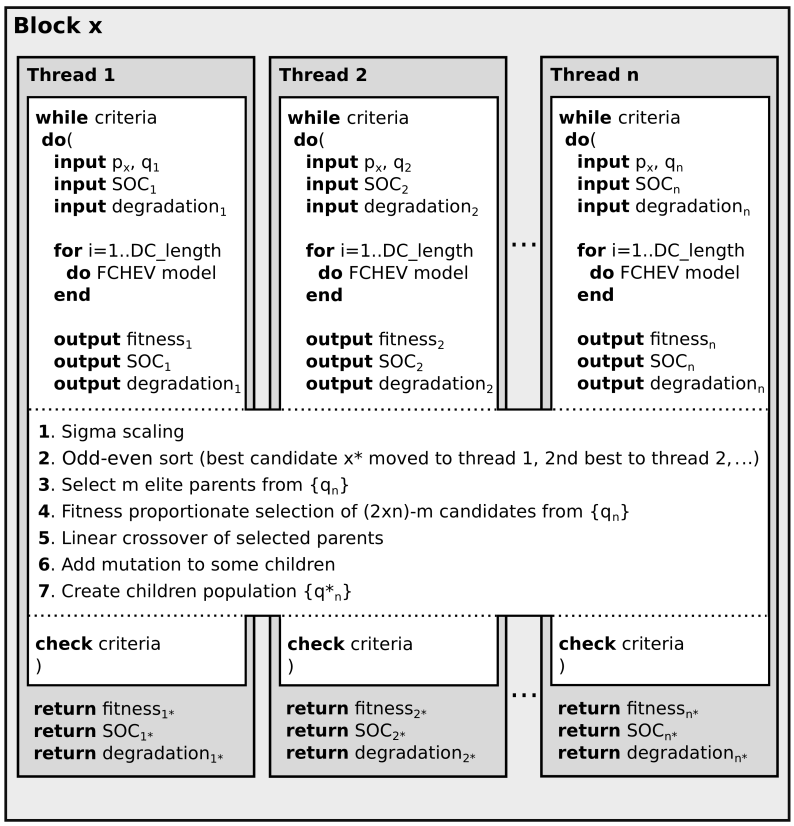


Figure 4.14. Thread synchronization for the kernel within one block

tasks. The GA, on the other hand, is mostly a *non-embarrassingly parallel* workload. Thread communication is required for many of its operators.

The kernel is repeatedly executed until the GA converges. Each thread of one block gets assigned the same design vector p but different candidates for q. The battery's state-of-degradation and SOC vary for each thread and are therefore quantified by independent state variables. The fact that threads are grouped in warps and are not necessarily synchronized needs to be considered during the implementation of the GA. Within a warp, threads are automatically synchronized. However, some GA operators require thread-communication between threads of different warps. It is therefore necessary to synchronize all threads before executing such operators. For example, the sigma scaling operator requires the computation of the average fitness of all threads in a block (Section 4.3.2). Hence, the operator can only be executed after all threads have finished the FCHEV simulation. Also, the average fitness is a mutual variable for all threads. Thus, it is sufficient to declare just one variable for the average fitness for each block. To avoid memory bank conflicts on the

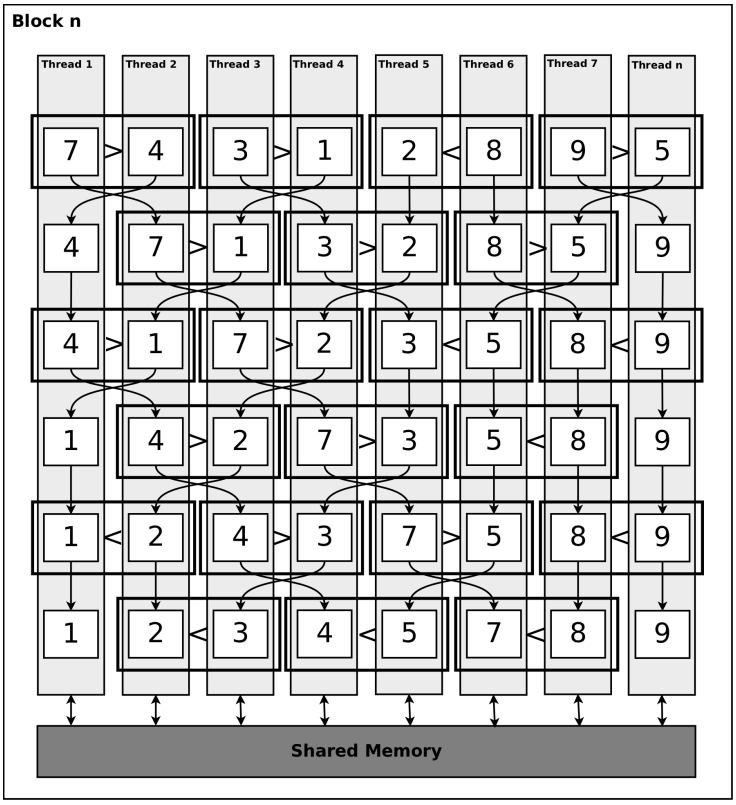


Figure 4.15. Parallel odd-even transposition sort implementation on the GPU architecture

GPU, it makes sense to have only one thread calculating the average fitness while all other threads are halted. A similar approach is used to determine the cumulative sum of the fitness proportionate selection operator: After synchronizing all threads in the block, all but one thread are paused. The still-active thread then determines the cumulative sums for all threads in the block.

The OETS is used to rank the candidates according to their fitness (Algorithm 4.3). Thread communication is limited to neighboring pairs of threads. The odd-even pairing of the threads alternates with each iteration of the sorting algorithm, as illustrated in Fig. 4.15. Once again, it is necessary to synchronize the threads if the total number of threads in the block exceeds the maximum warp size. Otherwise, parameters might swap their positions with parameters of different warps that might not be at the same stage of the algorithm at that time. The actual swapping command is invoked by only one of the threads in a pair. The other thread is paused in the meantime. If both threads would call the swapping function simultaneously, the parameters would be re-swapped to their original position. Hence, the function call would be redundant.

For the linear crossover of the design parameters, all threads have to be able to access all other threads in the block. Hence, thread synchronization must be performed before calling the crossover operator. Each thread creates offspring by calculating the mean of design parameters from the two selected threads. The threads return all state variables and the objective parameter to the host memory when the GA converges or its maximum number of iterations is reached.

Chapter 5

Performance Evaluation

This chapter investigates the impact of the GA parameter setup on the performance of the optimization methodology. First, the specifications of the hardware architectures used in this dissertation are presented. Next, the algorithm performances of the inner loop GA and the outer loop GA are studied as a function of their population sizes. Finally, the overall computation time of the optimization methodology is quantified and analyzed in regards to the CUDA programming model.

5.1 Hardware Configuration

To study the influence of the GPU hardware on the performance of the optimization methodology, the CUDA code is evaluated on three different GPU architectures. Table 5.1 provides an overview of the GPU systems used in this dissertation.

The consumer-level graphic card *GeForce GTX 460 2Win* is housed in a Dell T7500 workstation with 24GB (6x4GB) DDR-1333MHz RAM, Intel W5590 4-core processor (8M Cache, 3.33GHz), Dell 0D881F motherboard, and Nvidia Tesla C1060 GPU (4GB GDDR3 RAM) on 64-bit Ubuntu 11.04. This same workstation is used for the C/C++ and Matlab Simulink benchmarking in Section 5.3. The GPU Cluster incorporates 8 *Tesla M2050* computing units in a Dell server housing with 8 nodes connected via InfiniBand. Each M2050 processor board can be separately controlled.

Table 5.1. Three different GPU architectures used for code analysis and benchmarking

Specification	**GPU Architecture 1**	**GPU Architecture 2**	**GPU Architecture 3**
Category:	Gaming/consumer-level GPU	Computing Processor Board	GPU Cluster
Brand Name:	EVGA	NVidia	NVidia
Model:	GeForce GTX 460 2Win	Tesla M2050	Tesla M2050
Number of Devices:	2 x GeForce GTX 460	1	8 x Tesla M2050
Core Clock:	1350MHz	1150MHz	1150MHz
Memory Clock:	3600Mhz Effective	3092MHz	3092MHz
CUDA Cores:	2x336 (672)	448	8x448 (3584)
Streaming Multiprocessors:	2x7 (14)	14	8x14 (112)
Registers per Block:	16384	32768	32768
Memory Detail:	2x1024MB GDDR5	3072MB GDDR5	8x3072MB GDDR5
Memory Bandwidth:	2x115.2 GB/s	148.4 GB/s	8x148.4 GB/s
Processing Power:	2x907 (1814)GFLOPs	1040GFLOPs	8x1040 (8320)GFLOPs
Compute Capability:	2.1	2.0	2.0

5.2 Algorithm Performance

The population size has a significant impact on the performance of a genetic algorithm. The proposed optimization methodology of Section 4.4 contains two GAs, which are nested into each other. This integrated setup exponentially increases the influence of the population sizes on the overall performance of the optimization routine. In addition, the occupancy of the GPU is also dependent on the population size of the proposed optimization method. The occupancy is defined as the number of active warps over the number of maximum active warps, and thus is a function of the number of threads in a block (Section 4.5). The sub-population size of the inner optimization loop is equivalent to the number of threads in a block. Hence, the computational efficiency of the GPU is dependent on the population size.

The relationship between different population sizes and the performance of the proposed FCHEV optimization is investigated in the following sections. An estimation of the ideal population size for the inner loop GA is established in Section 5.2.1. The obtained results are then used in Section 5.2.2 to determine the optimal population sizes of the outer loop GA for various GPU architectures.

The drive cycle presented in Fig. 5.1 is used as a benchmark drive cycle for this performance study. The cycle consists of 9% stop-and-go, 38% urban, 17% suburban, 11% rural and

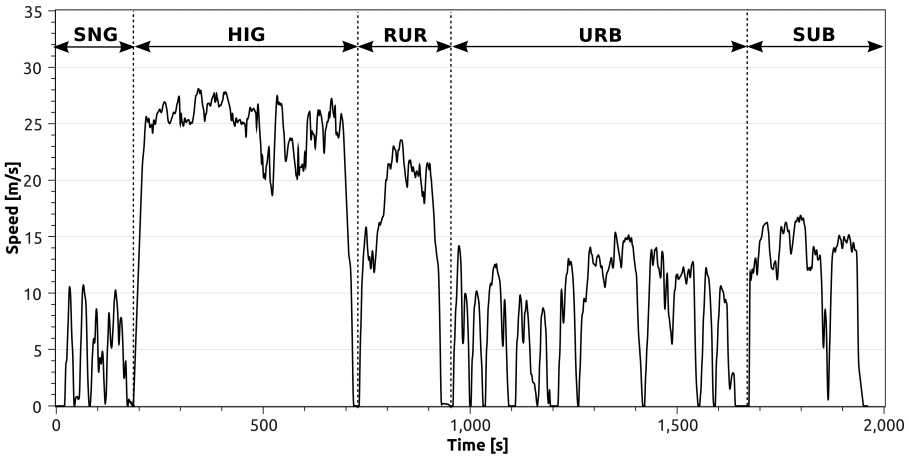


Figure 5.1. Benchmark drive cycle for population size optimization of the GAs. The percent composition in terms of driving scenarios is 9% SNG, 38% URB, 17% SUB, 11% RUR and 25% HIG.

25% highway driving. For lifetime simulations, the drive cycle is repeatedly used with the FCHEV model until the vehicle lifetime (100,000km) is reached. The initial conditions of the optimization problem are listed in Chapter 4.

5.2.1 Inner Loop GA

Convergence tests for the inner loop GA have been performed for 4 different population sizes (30, 50, 100, 200). The maximum possible population size is limited to 200 candidates due to memory constraints from the GPU architecture. Each series of tests consists of 5 simulations. The estimated convergence rate is determined from the average of the performed simulations. Fig. B.1-B.4 illustrate the convergence development for the four cases and Table 5.2 shows the obtained results.

Table 5.2. Inner loop GA: Average number of iterations until convergence for different population sizes

Population Size	30	50	100	200
Average number of iterations until convergence	17	8	4	3
Percentage improvement in convergence rate	-	10.6%	4%	1.3%

The convergence rate improves when the population size is increased, however, it suffers from diminishing returns. The larger the increase in population size, the smaller the improvement

in convergence rate. To determine the optimal population size, it is necessary to consider the kernel execution time, since it increases with the number of threads in a block. Therefore, the overall execution time of the inner loop is estimated as the product of the number of iterations until convergence and the kernel execution time. Fig. 5.2 shows the estimated GA execution time of the inner loop versus the inner loop population size. The kernel execution time is a function of the GPU occupancy, and thus increases in a step-like manner depending on the number of warps utilized for the block. All three GPU architectures in this study feature a maximum of 32 threads per warp if compiled with compute capability[1] 1.3. Hence, the optimal number of threads in a block is typically a multiple of 32. On account of this, the ideal population size of the inner loop GA is determined to be 192. Larger population sizes require the execution of an extra warp and therefore decrease the GPU occupancy, which outweighs the improved convergence rate.

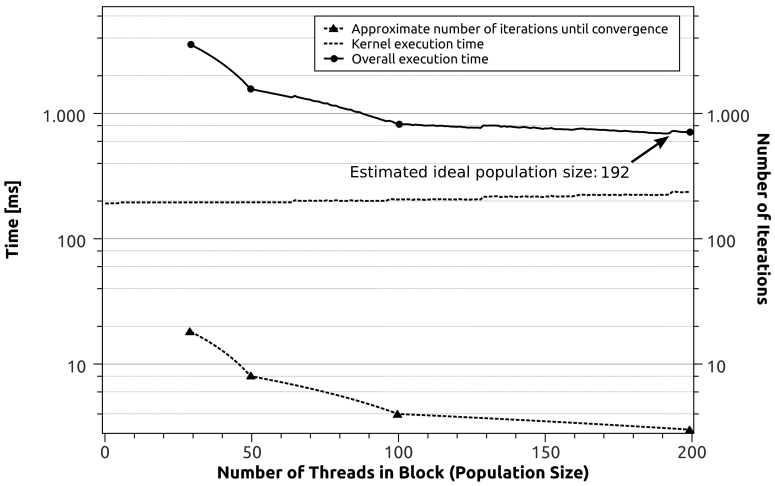


Figure 5.2. Estimation of the optimal population size for the inner loop GA on a GPU architecture

5.2.2 Outer Loop GA

An analogous approach is used to determine the optimal population size of the outer loop GA. First, convergence tests are performed for 8 different population sizes (20,30,50,80,100,200,400,800). The optimal population size of 192, which has been established in Section 5.2.1, is thereby used for the inner loop GA. Once again, each series of tests consists of 5 simulations, and the average of the test runs is used to determine the convergence rate. The results of the performed tests are illustrated in C.1-C.8, and the convergence rates are listed in Table 5.3.

[1]The compute capability describes the features supported by a CUDA-enabled device.

Table 5.3. Outer loop GA: Average number of iterations until convergence for different population sizes. *n/a* is used when no convergence is achieved.

Population Size	20	30	50	80	100	200	400	800
Average number of iterations until convergence	n/a	n/a	n/a	93	60	54	38	33

Again, the convergence rate improves substantially for smaller population sizes, but with diminishing returns as population size increases. Population sizes of less than or equal to 50 do not converge within 100 iterations. Once more, the kernel execution times for the different population sizes are needed to compute the overall execution time of the optimization. Since the kernel execution time of the outer loop is proportional to the number of drive cycles in the lifetime of the vehicle, the optimal population size can still be determined if the lifetime is reduced to just one drive cycle. Fig. 5.3 illustrates the kernel execution times for various population sizes when the vehicle lifetime is represented by the benchmark drive cycle.

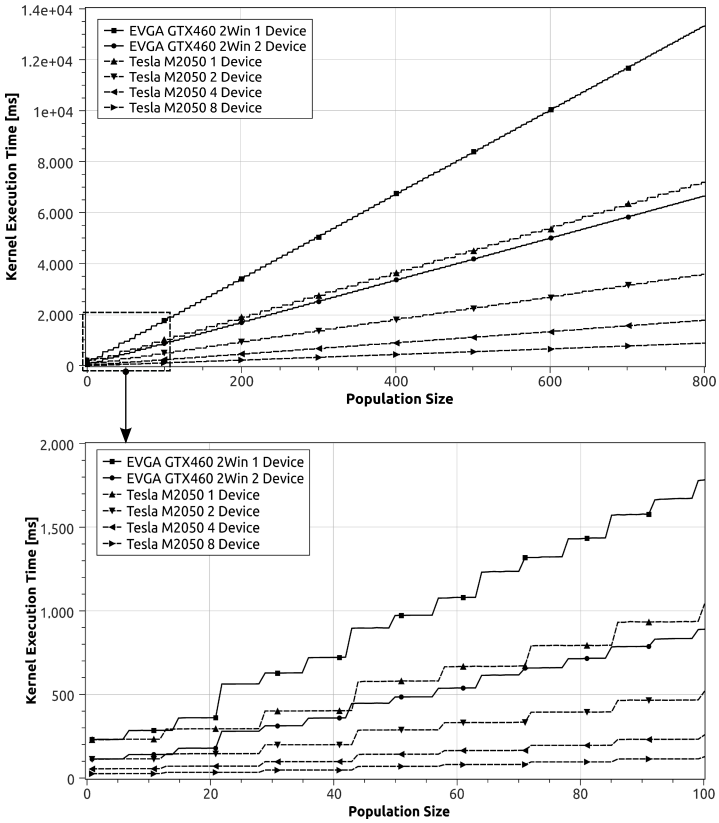


Figure 5.3. Outer loop GA: Kernel execution time versus population size and effect of SM occupancy.

On a larger scale, the kernel execution time is roughly proportional to the population size. However, upon closer examination, it is noticeable that it also exhibits step-like characteristics for smaller population sizes. While the kernel execution time rapidly increases for every 7 additional population candidates on the consumer-level GPU architecture (EVGA GTX460), it increases for every 14 candidates on the GPU cluster architecture (Tesla M2050). This phenomenon can be explained by the number of streaming multiprocessors (SMs) of each device. Consulting Table 5.1, each device of the consumer-level GPU consists of 7 SMs, and each device of the GPU cluster consists of 14 SMs. In the CUDA programming model, multiprocessors are grouped into SMs, which all execute the same instructions. When a CUDA program on the host CPU invokes a kernel grid, the blocks of the grid are enumerated and distributed to multiprocessors with available execution capacity [81]. If the number of blocks is not a multiple of the number of available SMs, then the execution capacity of the GPU is not fully utilized, hence the step-like characteristics of the kernel execution time.

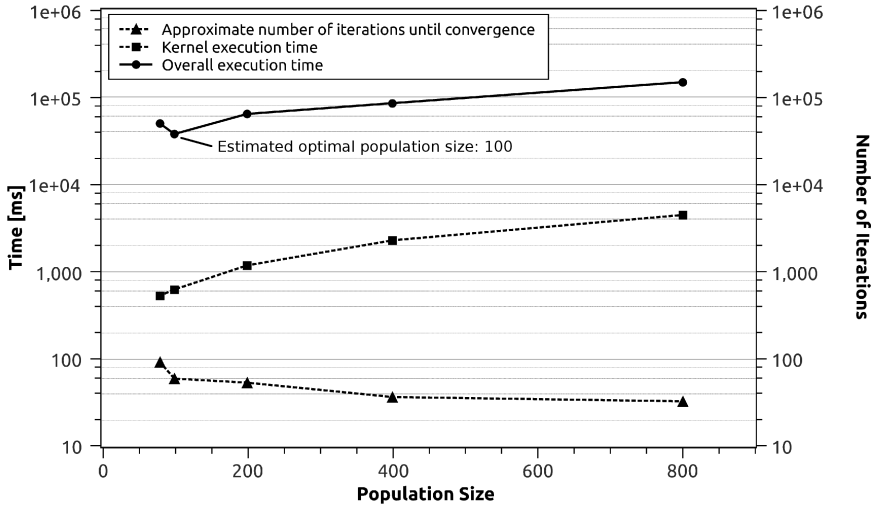


Figure 5.4. Estimation of the optimal population size for the outer loop GA on a 2xTesla M2050 GPU architecture

Fig. 5.4 shows the overall execution time of the optimization on the GPU cluster with two devices utilized. The estimated optimal population size that minimizes the overall computation time is determined to be 100. This result is also valid for the other GPU architectures, since the relationship between the kernel execution time and the population size is approximately linear (Fig. 5.3).

5.2.3 Overall Performance

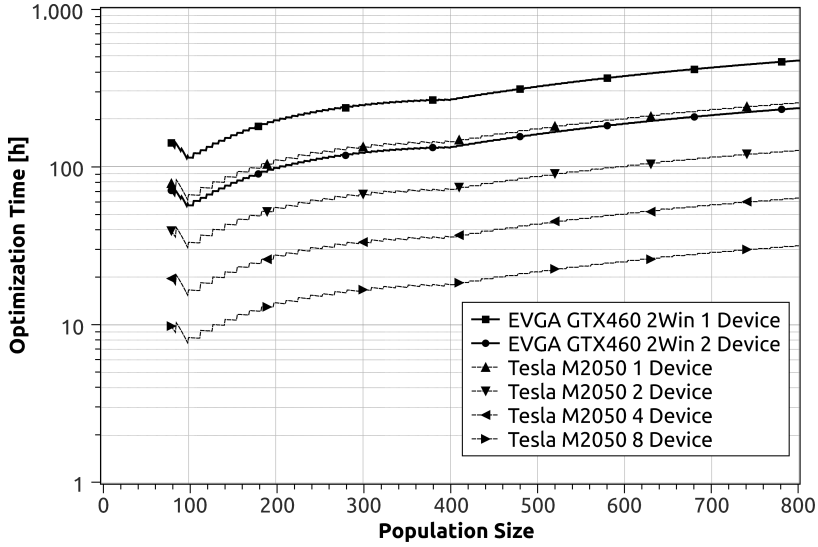
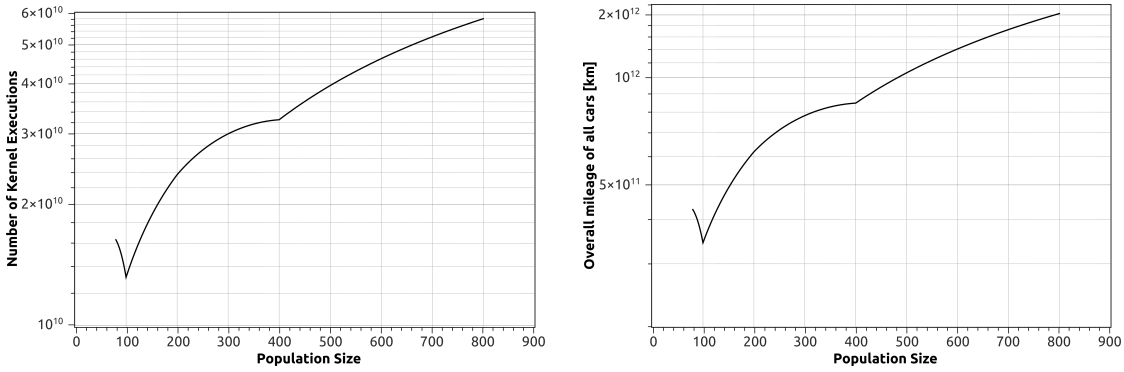


Figure 5.5. Overall optimization time for a vehicle lifetime of 100,000km on the consumer-level GPU architecture and the GPU cluster.

Considering the results of Section 5.2.1 and 5.2.2, it is now possible to estimate the overall optimization time for all architectures. The lifetime of the vehicle is set to 100,000km, and the benchmark drive cycle is repeatedly used until the lifetime is met. Fig. 5.5 shows the results for the GTX460 consumer-level graphics card, the single processing board and for the GPU cluster. The estimated optimization time for an ideally adjusted algorithm setup is approximately 7 hours on the GPU cluster when the computation load is distributed between its 8 devices, 55 hours on the 2 devices of the consumer-level GPU and 62 hours on the single processing unit.

The scalability of the optimization code on various GPU architectures is noticeable in the obtained results. Doubling the number of utilized devices within an architecture nearly halves the computation time of the optimization. This statement is valid for both the consumer-level card and the GPU cluster. The load balancing between all devices in an architecture is handled by the host. The invocation of the kernel and the memory transfer time are fast compared to the execution time of the kernel on the devices. As a consequence, the load balancing time is an insignificant fraction of the overall computation time. Hence, the CUDA code is fully scalable on multi-GPU systems.

It can be seen from Fig. 5.5 that the fully utilized consumer-level EVGA GTX460 2Win outperforms one Tesla M2050 device for all population sizes. The Tesla M2050 is a significantly more expensive computation board, which is especially designed for GPGPU applications. Looking



(a) Estimated number of kernel executions until convergence

(b) Estimated mileage simulated during the optimization until convergence

Figure 5.6. Estimated number of kernel executions and simulation performance for various population sizes.

at Table 5.1, the differences between the specifications of GPU architecture 1 and the specifications of one device of GPU architecture 2 are marginal. The largest discrepancies exist between the memory size and the memory bandwidth. Since both parameters are not the computational bottleneck for the FCHEV optimization, the consumer-level GPU delivers sufficient performance for the application.

Lastly, Fig. 5.6 demonstrates the magnitude of the computation in terms of kernels executed and in terms of the accumulated simulated mileage. The estimated number of drive cycles that are simulated on the GPU architecture during one optimization run is larger than $1.2 \cdot 10^{10}$. In other terms, the FCHEV model is executed at least 12 billion times during the optimization procedure. The sum of the simulated mileage for all vehicles thereby exceeds $3 \cdot 10^{11}$km. That is equivalent to 7.5 million trips around the world or 20 times the distance from the earth to the sun.

5.2.4 Section Findings

The choice of population size has a significant influence on the overall computation time. For example, selecting a population size of 200 for the outer loop GA nearly doubles the computation time compared to the optimal population size. If, in addition, the population size of the inner loop GA is set to 50, the overall optimization time will quadruple. This would extend the computation times on the GPU cluster to 28 hours and to an unreasonable 220 hours on the consumer-level GPU. Hence, small changes in the population size can have large effects on the convergence efficiency of the optimization routine.

Generally speaking, understanding the software-hardware interaction on GPU architectures is crucial knowledge for large-scale design optimization. Furthermore, a performance study for GA-based optimization methodologies is an essential procedure to fully utilize the hardware's capabilities, especially if the proposed methodology has a multi-level design.

In terms of hardware cost, cheaper consumer-level graphics cards deliver similar performance as more expensive GPGPU processing boards, as long as memory size and bandwidth are not a limitation.

5.3 Hardware Benchmarking

5.3.1 Computing Architecture

To benchmark the computation speed of the GPU architectures against the CPU implementations in C/C++ and Matlab/Simulink, the overall computation time, which is needed to execute a certain number of FCHEV simulations, is measured for each environment. For this, only the actual FCHEV simulation is evaluated, and thus the optimization algorithms are not part of the kernel. The benchmarking drive cycle of Section 5.2 is once again used as a reference, and the computation time is measured for various numbers of simulations. The following hardware/software environments are analyzed and compared in this study:

1. CPU architecture: 1xIntel W5590 3.33GHz, C/C++
2. CPU architecture: 1xIntel W5590 3.33GHz, Matlab/Simulink
3. CPU architecture: 4xIntel W5590 3.33GHz, Matlab/Simulink with Parallel Computation Toolbox
4. GPU architecture 1: 1xGeForce GTX460, CUDA
5. GPU architecture 1: 2xGeForce GTX460 (EVGA GTX460 2Win), CUDA
6. GPU architecture 2: 1xTesla M2050, CUDA
7. GPU architecture 3: 2xTesla M2050, CUDA
8. GPU architecture 3: 4xTesla M2050, CUDA
9. GPU architecture 3: 8xTesla M2050, CUDA

The kernel call of the C/C++ code is nested in a for-loop for this benchmarking study and is therefore subject to sequential execution. A similar setup is implemented for the Simulink code. A Matlab m-file contains a for-loop, which sequentially launches the Simulink FCHEV simulation. The design parameters and drive cycle data are allocated on the Matlab workspace, which is accessed by Simulink during the simulation. It is possible to add parallel execution capabilities to Simulink with Matlab's Parallel Computation Toolbox. The toolbox allows for concurrently executing a certain number of Simulink simulations corresponding to the number of available CPU cores. Since each CPU runs one simulation at a time, the maximum number of parallel simulations is limited to 4 in this study. In the following paragraphs, the computational performances of the three CPU implementations are analyzed and compared against the GPU implementations, which were introduced in Section 4. Fig. 5.7 illustrates the benchmarking results for the range of 1 to 100,000 performed simulations on each architecture.

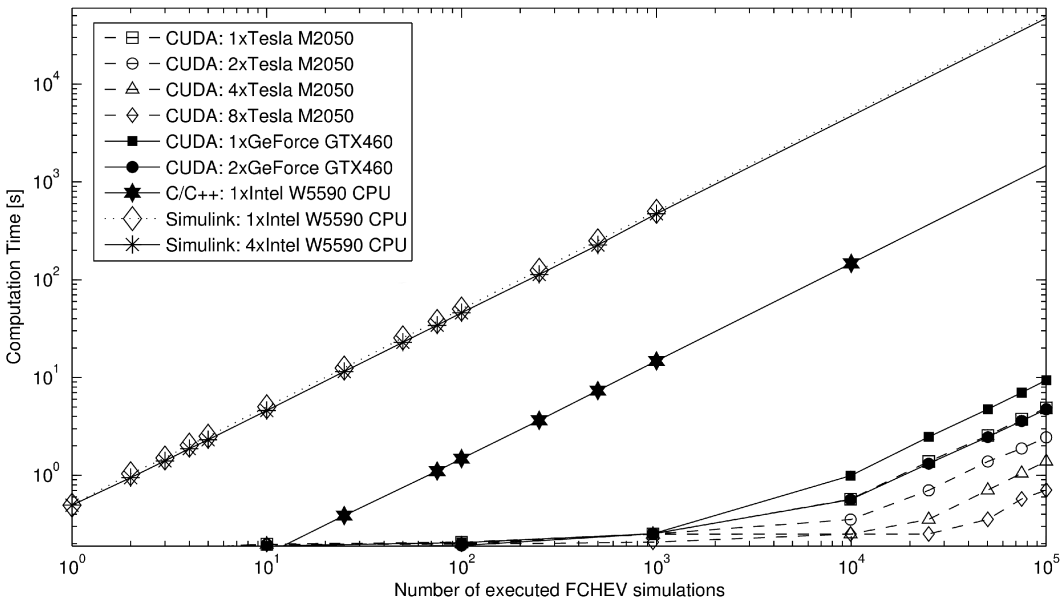


Figure 5.7. Benchmarking of 3 CPU-based FCHEV implementations in C/C++ and Simulink, and 6 CUDA-based simulations on various GPU architectures. For the GPU architectures, the number of threads in a block is set to 200 if the number of simulations is larger than or equal to 200.

It is evident that the GPU-based implementations perform significantly better than the CPU-based implementations for the entire abscissa range. The larger the number of executed simulations, the greater the advantages of the parallel GPU architecture. On the contrary, CPU-based simulations do not benefit from larger simulation numbers due to their sequential nature. Hence, the simulation time is approximately linear to the number of executed simulations. The graphical Mat-

lab/Simulink environment by far requires the longest computation times of all tested environments. This is because Simulink offers numerous graphical and functional features, which are computationally expensive, even if not utilized. Moreover, parallel computation with Simulink using the Matlab Parallel Computation Toolbox only improves the single-threaded implementation by less than 10% and is therefore inefficient. The C/C++ implementation, on the other hand, is remarkably more efficient than the Simulink implementations, but yet well behind the GPU simulations in terms of computation time. Looking at Table 5.4, the speedup of the C/C++ code compared to Simulink is larger than 30x for 100,000 performed FCHEV simulations. However, the slowest of the tested GPU architectures still outperforms the C/C++ code by 164x, and the fully utilized GPU cluster is a noteworthy 2190x faster. In overall computation time, the Simulink environment takes 40,000s (11.1h) to evaluate 100,000 simulations, the C/C++ code runs for 1,500s (25min), and the fastest GPU architecture fulfills the same task in 0.7s. In the previous section, the minimum number of kernel executions until convergence of the optimization routine is estimated to be $1.2 \cdot 10^{10}$. Considering the discussed results, the overall optimization time for the Simulink environment can be assumed to exceed 150 years, and the C/C++ implementation might take longer than 5 years to converge. From a practical point of view, computation times of that magnitude are infeasible and therefore require simplifications. Fig. 5.8 shows a closer view of the benchmarking results for the proposed GPU architectures.

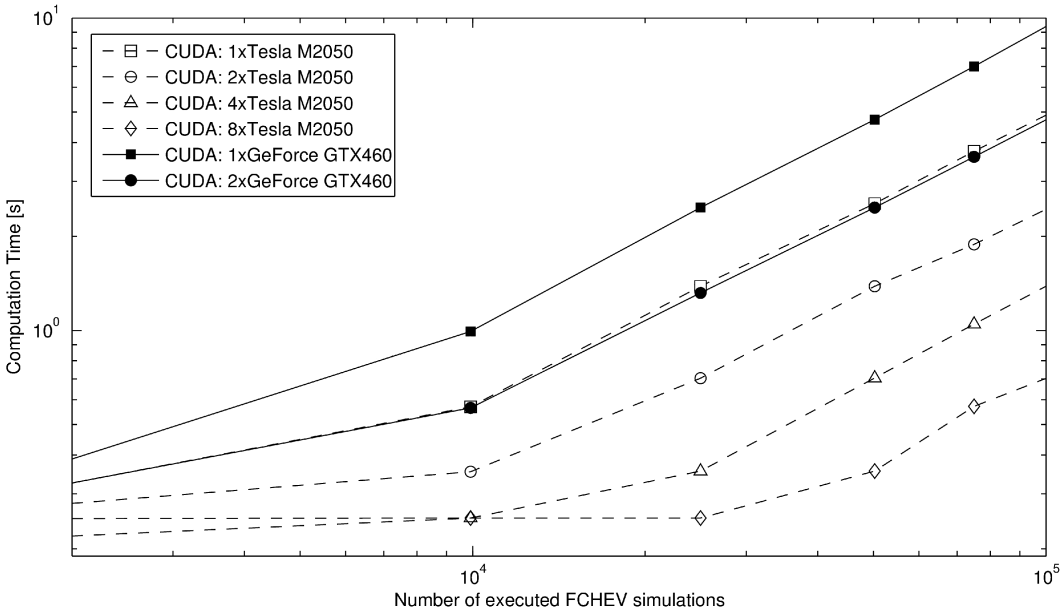
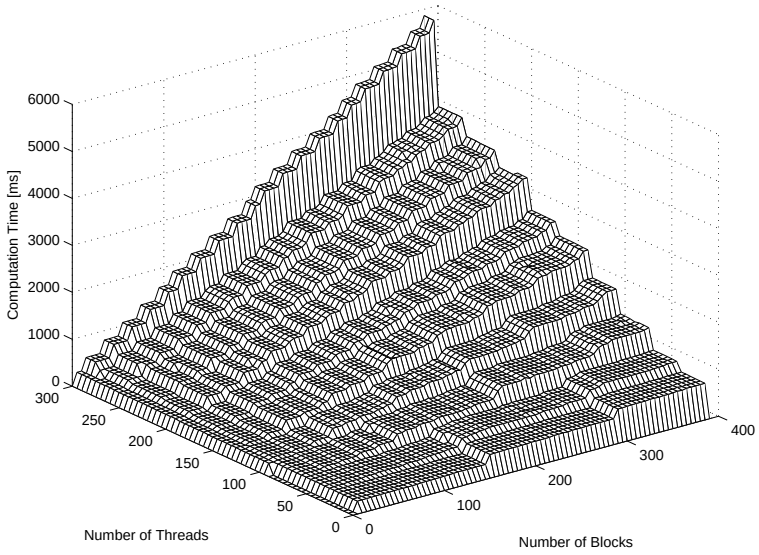


Figure 5.8. Benchmarking of 6 CUDA-based GPU environments for 1,000-100,000 executed simulations. The number of threads in a block is set to 200.

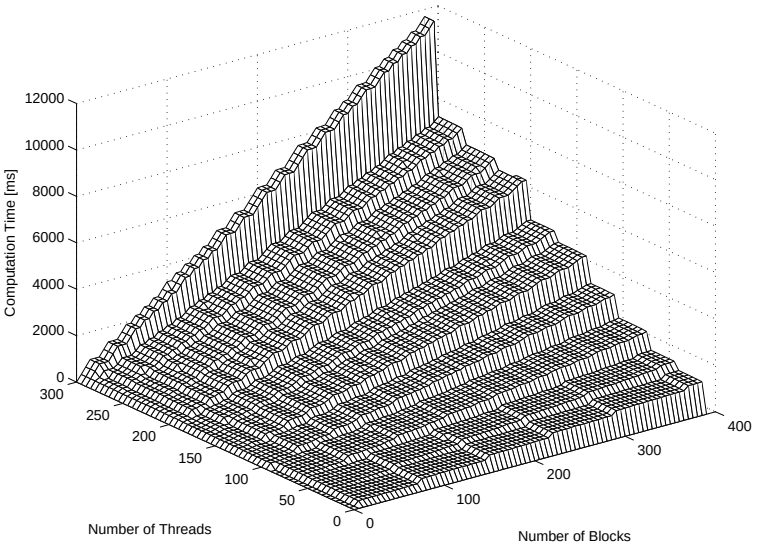
It is visible that all of the tested GPUs need a very high number of parallel tasks to reach their maximum efficiency. The increase in computation time is approximately proportional to the increase in number of simulations if more than 10,000 simulations are performed on the EVGA GTX460 2Win GPU and if more than 80,000 simulation are performed on the fully utilized GPU cluster (8 active devices). The measured computation time for 100,000 simulations ranges between 0.7s for the GPU cluster and 10s for one GTX460 device of the EVGA unit. Referring to Table 5.4, the computation speed within one GPU architecture increases proportionally to the number of utilized devices. For example, a 1.99x speedup is measured between utilizing only one GTX460 device of the EVGA unit and using both devices. The speedup factor when doubling the number of utilized devices on the GPU cluster is slightly smaller (1.92x, 1.83x, 1.97x), which can be explained by the difference in hardware configuration of the two GPU architectures. While the two devices of the consumer-level graphics card are incorporated into one GPU unit, which is connected to the motherboard via one PCI x16 slot, each GPU unit of the cluster is allocated to a separate CPU node. Hence, the increased communication latency reduces the computation speed .

Table 5.4. Speedup comparison matrix for 100,000 performed FCHEV simulations. Speedup factor is obtained as column value divided by row value.

	Simulink: 1xIntel W5590 CPU	Simulink: 4xIntel W5590 CPU	C/C++: 1xIntel W5590 CPU	CUDA: 1xTesla M2050	CUDA: 2xTesla M2050	CUDA: 4xTesla M2050	CUDA: 8xTesla M2050	CUDA: 1xGeForce GTX460	CUDA: 2xGeForce GTX460
Simulink: 1xIntel W5590 CPU	$1\times$	$1.06\times$	$33.81\times$	$10728\times$	$20660\times$	$37898\times$	$74774\times$	$5596\times$	$11129\times$
Simulink: 4xIntel W5590 CPU	$0.94\times$	$1\times$	$31.91\times$	$10150\times$	$19499\times$	$35770\times$	$70574\times$	$5282\times$	$10504\times$
C/C++: 1xIntel W5590 CPU	$0.03\times$	$0.03\times$	$1\times$	$315\times$	$605\times$	$1110\times$	$2190\times$	$164\times$	$326\times$
CUDA: 1xTesla M2050	$9.3 \cdot 10^{-5}\times$	$9.8 \cdot 10^{-5}\times$	$3.2 \cdot 10^{-3}\times$	$1\times$	$1.92\times$	$3.52\times$	$6.95\times$	$0.52\times$	$1.03\times$
CUDA: 2xTesla M2050	$4.8 \cdot 10^{-5}\times$	$5.1 \cdot 10^{-5}\times$	$1.6 \cdot 10^{-3}\times$	$0.52\times$	$1\times$	$1.83\times$	$3.62\times$	$0.27\times$	$0.54\times$
CUDA: 4xTesla M2050	$2.6 \cdot 10^{-5}\times$	$2.8 \cdot 10^{-5}\times$	$9.0 \cdot 10^{-4}\times$	$0.28\times$	$0.55\times$	$1\times$	$1.97\times$	$0.15\times$	$0.29\times$
CUDA: 8xTesla M2050	$1.3 \cdot 10^{-5}\times$	$1.4 \cdot 10^{-5}\times$	$4.6 \cdot 10^{-4}\times$	$0.14\times$	$0.28\times$	$0.50\times$	$1\times$	$0.07\times$	$0.15\times$
CUDA: 1xGeForce GTX460	$1.8 \cdot 10^{-4}\times$	$1.9 \cdot 10^{-4}\times$	$6.1 \cdot 10^{-3}\times$	$1.92\times$	$3.69\times$	$6.77\times$	$13.36\times$	$1\times$	1.99
CUDA: 2xGeForce GTX460	$8.9 \cdot 10^{-5}\times$	$9.5 \cdot 10^{-5}\times$	$3.1 \cdot 10^{-3}\times$	$0.97\times$	$1.85\times$	$3.41\times$	$6.7\times$	$0.50\times$	$1\times$



(a) Tesla M2050



(b) GeForce GTX460

Figure 5.9. Computation times for different degrees of utilization of the GPU architecture. The FCHEV simulation for the benchmarking drive cycle is used as a reference kernel.

5.3.2 Device architecture

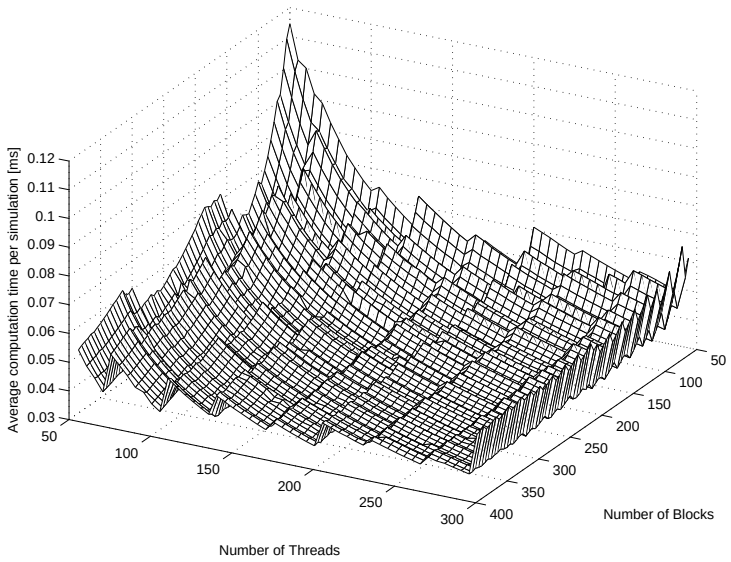
The computation speed of the Tesla M2050 device and the GeForce GTX460 device are investigated for various degrees of utilization in the following paragraphs. As mentioned in Section

5.2.1, the performance of a GPU is dependent on the design and the exploitation of the memory hierarchy of the CUDA programming model. Therefore, each GPU device has an ideal mode of utilization for a specific application. Fig. 5.9 illustrates the computation times of both discussed devices for various architecture utilizations. The FCHEV simulation for the benchmarking drive cycle is thereby used a reference kernel.

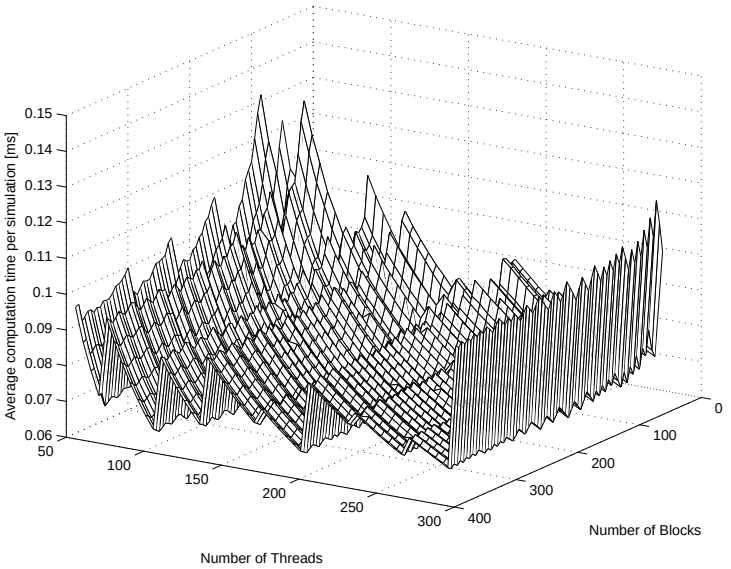
First, it is visible that the average computation time for all degrees of utilization is roughly twice as fast for the Tesla M2050 device. Secondly, both graphs exhibit plateaus of similar computation times but differing degrees of utilization. For both devices, the plateaus increase every 32 threads, which is equivalent to the maximum warp size of both units. If the number of threads in a block is not a multitude of 32, the maximum occupancy of the GPU is not fully utilized. While the width of the plateaus is identical for both devices, the lengths in terms of number of blocks is notably different. Generally, the plateau length decreases for larger numbers of threads in a block. The average plateau length of the Tesla M2050 GPU is approximately twice as long as the average plateau length of the GeForce GTX460 GPU. This is due to the execution capacities of both devices, since the Tesla M2050 incorporates 14 SMs and the GeForce GTX460 7 SMs. And thirdly, if the number of threads in a block is larger than 290, the computation times are disproportionately higher. This steep increase can be explained by the memory model of the GPU architecture. Automatic variables declared in a kernel reside in registers, which provide very fast access. However, the register space is limited for each SM. The larger the number of threads in a block, the smaller the number of registers available for each thread. If the register memory is exhausted, additional variables will instead be allocated to the local memory of the GPU. Accessing local memory is significantly slower, and the resulting memory latency therefore increases the overall computation time.

The average computation times per simulation for different degrees of utilizations are presented in Fig. 5.10. Choosing larger block sizes generally increases the overall computation speed per simulation. On the other hand, determining the optimal number of threads is not as trivial. The average computation time fluctuates like a sawtooth wave when increasing the number of threads. The computation time thereby peaks every 32 threads for both devices corresponding to their maximum warp size. However, the amplitude of this fluctuation is significantly larger for the GeForce GTX460 since it incorporates less SMs to leverage the workload. Hence, achieving good occupation levels is even more crucial for the consumer-level GPU. Furthermore, the GTX460 possesses less registers per block. Larger numbers of threads in a block therefore have a greater

influence on the memory latency caused by the need to access the local memory than for the Tesla M2050 GPU.



(a) Tesla M2050



(b) GeForce GTX460

Figure 5.10. Average computation time per kernel for different degrees of utilization of the GPU architecture. The FCHEV simulation for the benchmarking drive cycle is used as a reference kernel.

Chapter 6

Optimization Results

This chapter discusses results of the proposed optimization methodology for various objectives and is broken down into three sections. The first section demonstrates the applicability of the DCGT for large-scale optimization. A driving profile, which has been created from recorded driving data, is assigned to this DCGT. The optimization results for the DCGT-generated data are then compared to the optimization results for the recorded data. Finally, advantages of DCGT-based optimization over conventional optimization methods are identified and studied. In the second section, a sensitivity analysis is performed to investigate how small changes in driving style impact the outcome of an optimization. Next, optimal designs for various driving profiles and various levels of driver aggressiveness are determined and presented. The efficiency gains, in terms of fuel savings, are then computed for each designs relative to non-optimal designs. The last section examines the influence of parameter uncertainties on the optimization results using the example of battery degradation.

6.1 DCGT-Based Optimization

The FCHEV optimization is performed for three cases to investigate if the DCGT is applicable to large-scale EV design optimization and to study its possible advantages over conventional optimization methodologies.

- **Case 1:** 10h of driving data recorded in the City and County of Honolulu (9% SNG, 38% URB, 17% SUB, 11% RUR and 25% HIG): The drive cycles of the data set are sequentially and repeatedly utilized by the inner loop of the optimization routine until the lifetime convergence criteria of 100,000km is met.

- **Case 2:** A stochastic driving profile created by analyzing the data set of case 1 according to the proposed methodology of Sect. 3.1: The obtained stochastic parameters are implemented into the DCGT. The DCGT generates one stochastic drive cycle for each iteration of the inner optimization loop until the overall mileage of all cycles exceeds 100,000km.

- **Case 3:** A conventional approach of utilizing just one representative drive cycle: As described by Austin et al. [38], characteristic driving pulses are selected from the recorded data set and assembled to form a representative drive cycle as illustrated in Fig. 5.1. The drive cycle consists of 2000 seconds of drive cycle data with an equivalent composition in terms of driving scenarios to that of the recorded data set. The drive cycle is repeatedly used in the inner loop of the optimization procedure until the overall mileage is larger than 100,000km.

Table 6.1. Results of the design optimization for cases 1-3 and average fuel consumption when the obtained results are tested for the three different sets of driving data: ((a) 10h of recorded real-world driving data, (b) 10h of DCGT generated drive cycles, (c) one representative cycle created from (a) according to Austin et al. [38])

Design Parameter	**Case 1**	**Case 2**	**Case 3**
Max. Motor Power	62kW	63kW	48kW
Number of Fuel Cells	331	334	231
Battery Pack Capacity	6.3Ah	6.4Ah	4.7Ah

Avg. H_2 consumption for objective cycle data of:	**Case 1**	**Case 2**	**Case 3**
Case 1: 10h recorded data	$3.87\frac{g}{km}$	$3.89\frac{g}{km}$	$3.86\frac{g}{km}$ [2]
Case 2: 10h DCGT data	$3.81\frac{g}{km}$	$3.83\frac{g}{km}$	$3.79\frac{g}{km}$ [2]
Case 3: Rep. cycle (Fig.5.1)	$3.95\frac{g}{km}$	$3.98\frac{g}{km}$	$3.75\ \frac{g}{km}$

Table 6.1 shows the optimal component design for all three scenarios. The proposed designs for cases 1 and 2 are almost identical (x_1=[84hp, 331, 6.3kWh], x_2=[86hp, 334, 6.4kWh]), which indicates that the DCGT provides a very good representation of the original driving profile in all aspects. The DCGT power train design (case 2) is slightly more powerful for all three design parameters, which might be caused by high-frequency velocity noise at very high vehicle speeds, that are not sharply constrained by the DCGT (Fig. 3.6).

[2] The max. motor power is smaller than some of the power demand peaks. Therefore, the power demand was constraint to the max. motor power, which distorts the result by slightly reducing the average fuel consumption.

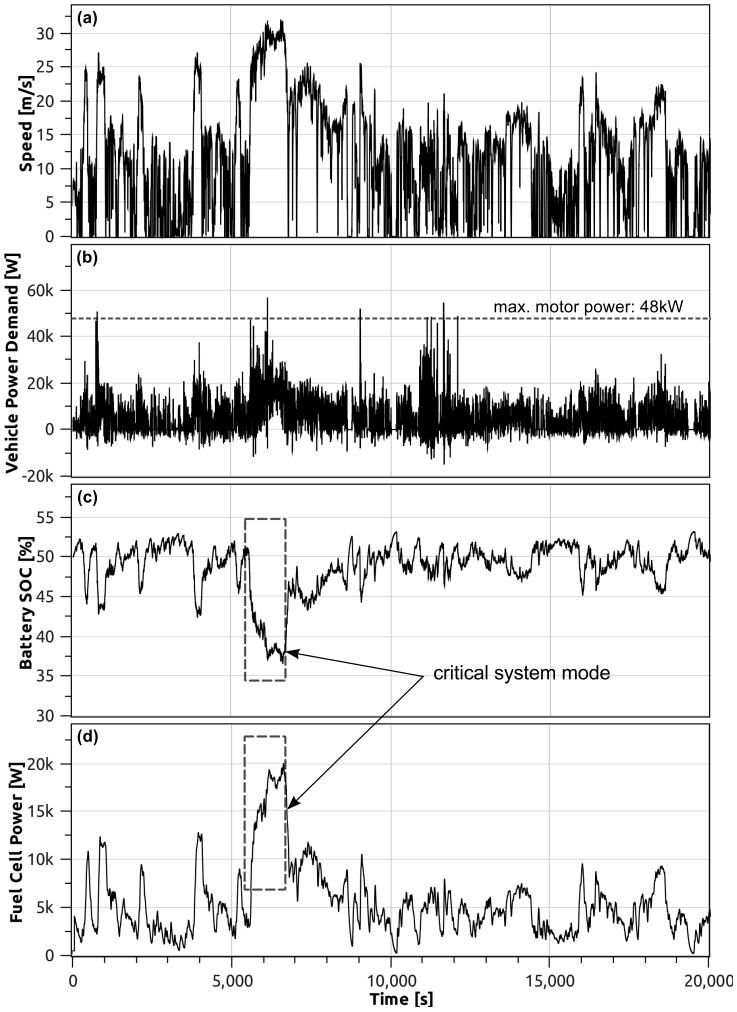


Figure 6.1. Case 3 optimization results tested with 20,000s of recorded drive cycle data. (a) Recorded drive cycle displayed as speed vs time. (b) Duty cycle of the proposed design. The power demand exceeds the maximum power of the proposed motor at several occasions. (c) Battery state-of-charge (d) Fuel cell stack power demand.

The second part of the table lists the average fuel consumption for the three cases when applied to each of the drive cycle data sets. As expected, the average H_2 consumptions for cases 1 and 2 are very similar. Case 2 has a marginally higher consumption for all three data sets which can be explained by the increased weight of the vehicle due to the larger power train.

Looking at the results of case 3, one could assume that the proposed design is superior to the results of cases 1 and 2. The case 3 power train is significantly smaller sized than its competitors which leads to drastically reduced fuel consumption for case 3 ($3.75\frac{g}{km}$ vs. $3.95\frac{g}{km}$ and $3.98\frac{g}{km}$) and a slightly reduced fuel consumption for case 1 and case 2. However, the best solution is not

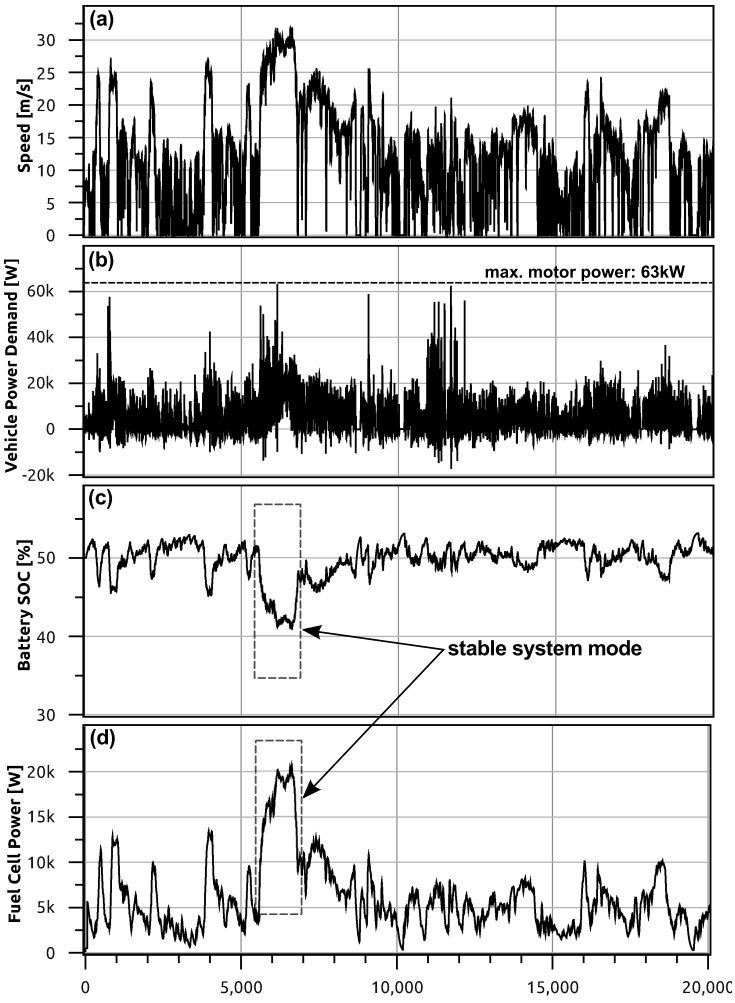


Figure 6.2. Case 2 (DCGT) optimization results tested with 20,000s of recorded drive cycle data. (a) Recorded drive cycle displayed as speed vs time. (b) Duty cycle of the proposed design. The power demand does not exceed the maximum power of the proposed motor. (c) Battery state-of-charge (d) Fuel cell stack power demand.

necessarily the solution that achieves the best fuel economy, but rather the one the achieves the best fuel economy while also completely satisfying the power demand and providing stable operation at all times.

Fig. 6.1 provides performance data of case 3 when executed for 20,000s of recorded drive cycle data (objective of case 1). Evidently, this motor design is insufficiently dimensioned for certain power demand peaks of the duty cycle which exceed 48kW (Fig. 6.1(b)). Due to its relatively small occurrence rate, this phenomenon cannot entirely be accounted for by a single drive

cycle. The lack of available motor power might be a critical safety shortcoming in driving situations that require increased agility of the vehicle.

Fig. 6.1(c) and 6.1(d) show the battery state-of-charge and the power demand to the fuel cell stack. It is apparent that the battery and the fuel cell stack are not sufficiently sized for driving scenarios with an above average power demand, such as busy highway driving. The system enters a critical mode of operation when the power demand is too high for a long period of time. During this, the SOC drops rapidly to critical level, and the fuel cell stack is not powerful enough to compensate. Apparently, the case 3 design cannot maintain this unstable mode for a long period of time without reaching critical component levels, which might cause a sudden power-drop and possible damage to the system.

Conversely, Fig. 6.2 demonstrates that the described stability issues are taken into account when preforming a stochastic DCGT-based optimization. The motor (63kW) is sufficiently sized to satisfy the power demand even for rare, above-average power peaks. During busy highway driving, the SOC of the batter pack drops to a lower level, but the power loss is compensated for by the fuel cell stack. None of the system components reach critical levels.

6.2 Design Sensitivity Towards Peak Loads

All design parameters are restricted by functional constraints during the optimization routine. If a proposed design does not meet all constraints, it is eliminated as a possible solution. Some constraints are compulsory conditions that need to be fulfilled to guarantee stable operation of the system. Other constraints are subjective restrictions, which typically cannot be logically specified. For single-objective optimization methods, the subjective perception of the decision maker needs to be quantified in order to define a distinct constraint. The optimization procedure will then provide the best design solution for the choice of the decision maker. However, the decision maker cannot be certain that his constraint choice was ideal if he does not study the sensitivity between the constraint choice and the outcome of the optimization. Small changes in constraints might significantly better the fitness of the obtained solution. *Sensitivity Analysis* is a methodology that helps the decision maker understand the relationship between his choice and the optimization outcome, therefore enabling a better informed decision.

So far in this study, the functional constraints of the design parameters have been set to consistently fulfill the power demand for all possible scenarios in the lifetime of the vehicle. If just one time in its lifetime the vehicle cannot meet the demand for an above-average power peak, the

vehicle design is eliminated as a possible solution. However, if the corresponding peak demand was not caused by an essential driving maneuver, a potentially good design solution might be dismissed.

The following case study investigates the sensitivity of the optimal design towards changes in the intensity of power demand peaks. The basic driving style of the operator stays unchanged, but occasional, above-average power demands are reduced in their magnitude. The operator can then decide on the best trade-off between reducing the fuel consumption and decreasing the intensity of some exceptionally demanding driving maneuvers.

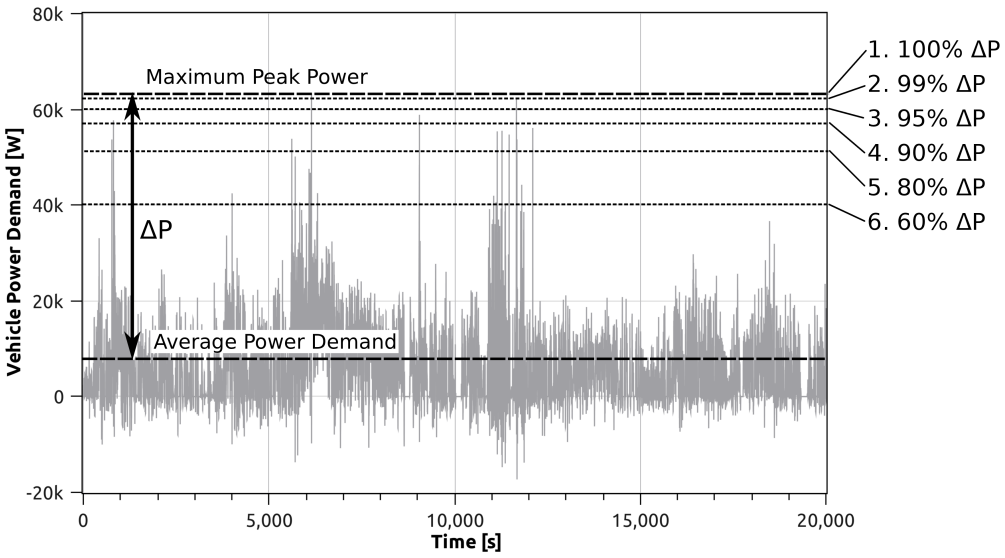


Figure 6.3. Sensitivity analysis for 6 different power peak characteristics. The magnitude of the power peak is determined as the difference between average power demand and maximum power demand.

Fig. 6.3 explains the approach of the proposed sensitivity study. The maximum power demand in the lifetime of the vehicle is used as a reference to define the maximum power intensity dP, which is defined as the maximum power demand in the lifetime of a vehicle minus the average power demand.

$$\Delta P(p,q) = \max_t P_d(t,p,q) - \overline{P_d(t,p,q)} \tag{6.2.1}$$

For the sensitivity study, an additional threshold constraint, which is based on dP, is introduced to restrict the power peaks. The following 6 constraints are investigated:

- Case 1: 100%dP
- Case 2: 99%dP

- Case 3: 95%dP
- Case 4: 90%dP
- Case 5: 80%dP
- Case 6: 60%dP

Since the power demand is a function of both the time and the design vectors p and q. dP is specific for each solution candidate. Furthermore, to constrain the power peaks, dP needs to be known from the beginning of the fitness evaluation. Therefore, the maximum and average power demand need to be determined for each solution candidate during an a-priori lifetime simulation before dP can be derived. The fitness evaluation is then executed one more time with the additional threshold constraint applied.

The objective and methodology of case 2 in Section 6.1 is once again employed for this analysis.

Table 6.2. Optimal design, fuel consumption, and improvement in fuel consumption for cases 1-6 of the sensitivity study.

Design Parameter	**Case 1: 100%ΔP**	**Case 2: =99%ΔP**	**Case 3: 95%ΔP**	**Case 4: 90%ΔP**	**Case 5: 80%ΔP**	**Case 6: 60%ΔP**
Max. Motor Power [kW]	63	62	60	56	51	40
Number of Fuel Cells	334	331	329	323	317	302
Battery Pack Capacity [Ah]	6.4	6.3	6.2	6.1	5.9	5.7
Avg. H_2 consumption [g/km]	3.86	3.85	3.83	3.79	3.71	3.54
Improvement in fuel consumption [%]	0	0.3	0.8	1.8	4.0	9.0

Table 6.2 presents the results for the 6 evaluated cases. Most noticeably, the maximum motor power decreases proportionally with the reduction of dP. Since dP is relatively large compared to the average power demand, the motor efficiency improves when decreasing the motor size (Fig. 2.5). Therefore, the maximum rated load of the motor is roughly equivalent to the maximum power demand in this example.

Furthermore, reducing the motor size decreases the weight of the power train, which in turn abates the power demand. As a result, the power demands to the fuel cell stack and battery pack decrease accordingly, which affects their optimal sizing. The optimal battery capacity and the optimal number of fuel cells both shrink by 12% if the maximum motor power is constrained to 60%dP. Overall, the improvement in fuel consumption is insignificant for small changes in dP.

However, the fuel efficiency increases by 4% and 9% for 80%dP and 60%dP, respectively. While efficiency gains of that magnitude are substantial, they come with noticeable restrictions on the operator's driving style.

6.3 Application and Driver-Specific Vehicle Design

One of the major advantages of the proposed optimization methodology is that a vehicle can be optimized for specific applications and driving styles. Generally, commercial vehicles are designed for a variety of driving scenarios, whereby some cars might be more suitable for certain driving environments than others. For example, smaller cars might be more efficient in busy city traffic, while larger cars are usually more advantageous for fast highway driving. Nevertheless, smaller cars typically feature a high enough level of motorization, which makes them suitable for long distance highway driving. Thus, they can be considered all-purpose vehicles with a preferred application.

An all-purpose vehicle might perform well in terms of fuel efficiency when used for a variety of driving scenarios, but might be sub-optimal if the application range is limited. Application-specific designs might be the better choice for driving profiles that are restricted in terms of their driving scenarios. For example, in metropolitan areas, some vehicles might primarily be operated in SNG and URB driving conditions. Knowing the cost of the option to possibly use the vehicle for more demanding driving scenarios is of interest, since it might have an impact on the driver's choice of vehicle type.

Personal driving style is the second major influence on the fuel consumption of the vehicle. Aggressive drivers accelerate harder on a more frequent basis and tend to drive faster. Vehicles for aggressive drivers need to be capable of providing extra power for rapid maneuvers even in demanding situations, such as fast highway driving. For drivers with lower levels of aggressiveness, smaller power trains might be sufficient and thus are more efficient. The driver's *level of aggressiveness* is classified into three categories in this study:

- **LOW** (below-average acceleration, decelleration, cruising speed, and velocity noise frequencies)
- **MEDIUM** (average acceleration, decelleration, cruising speed, and velocity noise frequencies)

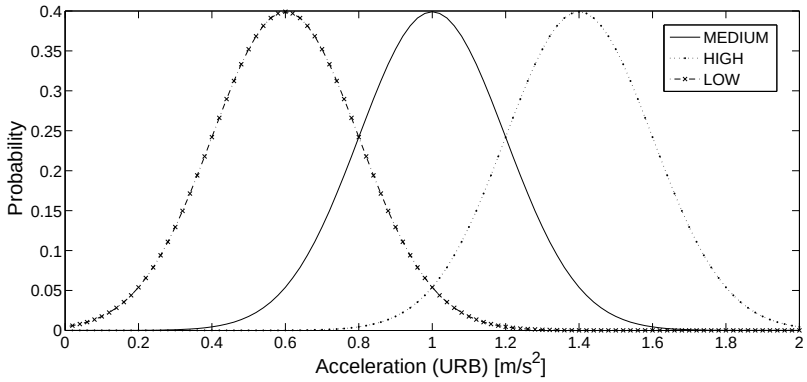


Figure 6.4. Probability function for acceleration in an urban environment for three different levels of aggressiveness (low, medium, and high)

- **HIGH** (above-average acceleration, decelleration, cruising speed, and velocity noise frequencies)

Fig. 6.4 explains how different levels of aggressiveness are implemented into the DCGT on an example of the acceleration probability during urban driving. The probability functions for acceleration, deceleration, cruising speed, and velocity noise are the key functions for the characterization of a driving style. The probability curves are shifted along the positive direction of the abscissa to increase the level of aggressiveness, and against it to decrease the level of aggressiveness.

In this section, the optimal power train designs for five different driving profiles are compared. Four of the investigated profiles are application-specific profiles, which are limited to certain types of driving scenarios. The fifth profile is an all-purpose profile, that incorporates all possible driving scenarios. Three different levels of driver aggressiveness are assigned to each of the five driving profiles to study the impacts of driving styles on the vehicle's design.

- **Case 1:** Vehicle operated solely in busy stop-and-go traffic: 100% SNG
- **Case 2:** Vehicle operated in stop-and-go traffic and normal urban traffic: 50% SNG, 50% URB
- **Case 3:** Vehicle operated solely in non-busy urban traffic: 100% URB
- **Case 4:** Vehicle used solely for highway driving: 100% HIG
- **Case 5:** All-purpose vehicle equally used for all possible driving scenarios: 20% SNG, 20% URB, 20% SUB, 20% RUR, 20% HIG

In total, 15 driving profiles (5 cases x 3 level of aggressiveness) are created and applied to the DCGT. The proposed optimization routine is then executed for each profile with a vehicle lifetime of 100,000km. Table 6.3 lists the optimal component sizes and fuel consumptions for each driving profile.

Table 6.3. Optimal component sizing and fuel consumption for various driving profiles and levels of driver aggressiveness

Level of Driver Aggressiveness	**Design Parameter & Fitness**	100% SNG	50% SNG 50% URB	100% URB	100% HIG	20% SNG/URB/ SUB/RUR/HIG
LOW	Max. Motor Power [kW]	14	14	14	27	26
	Number of Fuel Cells	88	130	116	254	253
	Battery Pack Capacity[Ah]	3.17	3.46	3.81	5.56	5.34
	Avg. H_2 consumption [g/km]	0.75	0.73	0.72	2.11	1.52
MEDIUM	Max. Motor Power [kW]	14	15	15	42	39
	Number of Fuel Cells	116	144	152	317	254
	Battery Pack Capacity [Ah]	4.37	4.81	4.86	6.73	6.82
	Avg. H_2 consumption [g/km]	1.11	1.07	1.04	2.99	2.25
HIGH	Max. Motor Power [kW]	29	29	28	83	77
	Number of Fuel Cells	163	186	172	480	364
	Battery Pack Capacity [Ah]	6.90	6.84	6.58	9.94	9.94
	Avg. H_2 consumption [g/km]	2.26	1.80	1.34	5.17	3.82

Several key findings can be drawn from the obtained results. First, the average fuel consumptions vary greatly between different applications ($0.75\frac{g}{km}..2.11\frac{g}{km}$) and different levels of aggressiveness ($0.75\frac{g}{km}..2.26\frac{g}{km}$). Note that an aggressive driver consumes almost three times as much fuel during SNG traffic as a driver with a low level of aggressiveness. Furthermore, HIG driving more than doubles the fuel consumption of a vehicle compared to URB driving. Interestingly, the fuel consumption rates for SNG driving and URB driving are of similar magnitude.

The second key finding is that the optimal component sizing significantly depends on the the application range of the vehicle. The driving profiles for cases 1, 2, and 3 have similar specifications in regards to motor power and battery capacity. However, the fuel cell stacks sizes are slightly smaller for pure SNG driving. This can be explained by the ratio of overall stopping time to overall driving time, which is typically higher for SNG traffic due to the increased number of stops. If the overall stopping time increases, the average power demand to the fuel cell stack decreases. In the proposed vehicle model, the fuel cell stack is utilized as a power source with

slow dynamic characteristics. Hence, the power demand to the fuel cell stack is similar to the average power demand of system, while the battery pack acts as a buffer. Since SNG driving has a high ratio of stopping time to driving time compared to urban driving, the power demand to the fuel cell stack is relatively small. The fuel cell stack can therefore be of smaller size. On the contrary, the case 4 and case 5 driving profiles require significantly more propulsion than the SNG and URB driving profiles. RUR and HIG driving distinguish themselves from inner-city driving through higher speeds and longer driving periods. In short, the vehicle operates mostly at high power levels during these driving scenarios. Additionally, driving maneuvers, such as quick accelerations, require exponentially more vehicle power if the vehicle speed is already high. Therefore, the higher the average speed during a driving scenario, the larger the necessity for a strong power train.

Third, the optimal component sizing has a considerably dependency on the driving style. The overall power demand increases exponentially with the acceleration rate. Aggressive drivers accelerate quicker and more frequently than drivers with lower levels of aggressiveness. Furthermore, the average cruising speed of a vehicle is typically higher for more aggressive driver. To combine both statements, an aggressive driver not only accelerates his vehicle faster than a non-aggressive driver, he also accelerates to higher speeds. Hence, aggressive acceleration is exponentially more demanding than mild acceleration. Accordingly, power trains must be exponentially more powerful for higher levels of driver aggressiveness.

The fourth finding concerns the relationship between the maximum motor power and the ratio between the average and the maximum speed of a driving scenario. looking at Fig. 2.5, the maximum motor efficiency lies between 50% and 80% of the rated load. If the vehicle is mainly operated at power levels that are significantly smaller than occasional power peaks, the optimal rated load of the motor is approximately equivalent to maximum power demand. The motor efficiency curve does not have a significant influence on the optimization in this case. However, if the power demand frequently reaches levels closer to the maximum power peak, the motor efficiency begins to have an influence on the optimization results. The motor size is then determined so that the average power demand of the driving scenario is close to the most efficient motor load. For example, case 4 naturally has a high ratio between average power demand and maximum power demand. For an aggressive driver, the ideal maximum motor power is determined to be 83kW. Case 5, on the other hand, incorporates the driving profile of case 4 with a share of 20%. The other 80% of its driving profile, however, decreases the ratio between average power and maximum power to a much lower level relative to case 4. Despite having to fulfill the same requirements during highway driving as

case 5, the optimal motor power size is determined to be 77kW, which is close to the maximum power peaks.

Lastly, the optimal capacity of the battery pack appears to be determined according to the most demanding driving scenario of a driving profile. For instance, the optimal battery capacities of case 5 are very close to those in case 4. Highway driving is certainly the most demanding scenario for an all-purpose vehicle. As mentioned before, the battery pack is used as an energy buffer in the power train and therefore serves mainly to compensate for transient power demand peaks. Since highway driving generates the largest fluctuations in power demand, the battery pack needs to be sufficiently dimensioned to account for these power oscillations.

6.3.1 Optimization efficiency

To examine the efficiency gains achieved by application and driver-specific optimizations, the obtained vehicle designs are cross-tested for all other driving profiles. In this section, each of the 15 results of Table 6.3 is tested for 100,000km of driving data for each of the 15 driving profiles, which were used to obtain the results. Table 6.4 shows the average fuel consumption and the difference as a percentage of fuel consumption between the optimal designs and the tested designs. Several interesting conclusions can be drawn from the results.

First, significant improvements in fuel consumption can be achieved if the power train design is optimized towards the application. For example, a vehicle, which is designed for aggressive highway driving, consumes nearly three times as much hydrogen during non-aggressive stop-and-go traffic than a vehicle specifically optimized for that purpose. Moreover, an all-purpose vehicle for aggressive drivers uses 2.5 times as much fuel during relaxed city driving than an optimized counterpart. Even within the same level of driver aggressiveness, the potential fuel savings are significant. If all-purpose vehicles are solely used for driving in an urban environment, they consume up to 60% more fuel than an optimized vehicle for the same driving style.

Secondly, individual drivers can achieve substantial fuel savings if they are willing to relax their driving style. As found before, the power demand, and thus the component sizing, increases exponentially with the level of aggressiveness. If an aggressive driver of an all-purpose vehicle reduces his level of aggressiveness from *high* to *low*, potential fuel savings of up to 55% are achievable using the same vehicle.

Thirdly, the drawback of optimizing a power train towards one application is that it might lead to a limited application range of the vehicle. This finding is illustrated in Table 6.4, where most entries of the lower triangular matrix are empty. Power trains, which are optimized for less demand-

ing driving profiles, are oftentimes not capable of fulfilling the requirements of more demanding driving profiles. Either the total power demand is larger than the maximum rated load of the motor, or the fuel cell stack and the battery pack are not capable of providing a sufficient power output over a required time range. In both cases, the power trains do not match the required functional constraints. Hence, if application flexibility is a design criterion for the optimization of a power train, the exact requirements need to be represented by the driving profile during the optimization.

Finally, a smaller and lighter power train does not necessarily lead to a more efficient vehicle. For example, the optimal motor and fuel cell stack size for all-purpose vehicles (case 5) is noticeably smaller than for highway vehicles (case 4), while their battery sizing is of a similar dimension. Nevertheless, if all-purpose vehicles are tested for case 4 driving profiles, the average fuel consumptions increase by up to 7%. The fuel cell stack and the motor are predominantly operated at non-optimal efficiency levels, which outweighs the advantages of reduced vehicle weight.

Table 6.4. Fuel efficiency matrix for the designs of Table 6.3 tested for 5 different driving profiles with 3 different levels of aggressiveness. The first line of each entry is the average fuel consumption in $\frac{g}{km}$, the second line is the difference in fuel consumption relative to the optimal design. If a tested design cannot fulfill the demand of the driving profile, its entry is set to "n/a".

Level of Driver Aggressiveness		**LOW**					**MEDIUM**					**HIGH**				
	FCHEV optimized for / **FCHEV tested for**	100% SNG	50% SNG 50% URB	100% URB	100% HIG	20%URB/SUB RUR/HIG	100% SNG	50% SNG 50% URB	100% URB	100% HIG	20%URB/SUB RUR/HIG	100% SNG	50% SNG 50% URB	100% URB	100% HIG	20%URB/SUB RUR/HIG
LOW	100% SNG	0.75 +0.00%	0.76 +1.33%	0.77 +2.67%	1.00 +33.33%	1.00 +33.33%	0.82 +9.33%	0.86 +14.67%	1.07 +42.67%	1.32 +76.00%	1.26 +68.00%	1.13 +50.67%	1.14 +52.00%	1.10 +46.67%	2.12 +182.67%	1.90 +153.33%
	50% SNG/ 50% URB	0.73 +0.00%	0.73 +0.00%	0.73 +0.00%	0.80 +9.59%	0.93 +27.40%	0.76 +4.11%	0.79 +8.22%	0.80 +9.59%	1.14 +56.16%	1.12 +53.42%	1.02 +39.73%	1.00 +36.99%	0.89 +21.92%	1.90 +160.27%	1.70 +132.88%
	100% URB	0.75 +4.17%	0.76 +5.56%	0.72 +0.00%	0.80 +11.11%	0.85 +18.06%	0.74 +2.78%	0.75 +4.17%	0.76 +5.56%	1.09 +51.39%	1.03 +43.06%	0.92 +27.78%	0.92 +27.78%	0.82 +13.89%	1.76 +144.44%	1.57 +118.06%
	100% HIG	n/a n/a	n/a n/a	n/a n/a	2.11 +0.00%	2.12 +0.47%	n/a n/a	n/a n/a	n/a n/a	2.34 +10.90%	2.28 +8.06%	2.27 +7.58%	2.28 +8.06%	2.21 +4.74%	2.95 +39.81%	2.75 +30.33%
	20% SNG/URB/SUB/RUR/HIG	n/a n/a	n/a n/a	n/a n/a	1.60 +5.26%	1.52 +0.00%	n/a n/a	n/a n/a	n/a n/a	1.65 +8.55%	1.73 +13.82%	1.67 +9.87%	1.68 +10.53%	n/a n/a	2.50 +64.47%	2.36 +55.26%
MEDIUM	100% SNG	1.24 +11.71%	1.19 +7.21%	1.14 +2.70%	1.60 +44.14%	1.28 +15.32%	1.11 +0.00%	1.13 +1.80%	1.15 +3.60%	1.58 +42.34%	1.51 +36.04%	1.35 +21.62%	1.35 +21.62%	1.25 +12.61%	2.44 +119.82%	2.18 +96.40%
	50% SNG/ 50% URB	1.18 +10.28%	1.12 +4.67%	1.08 +0.93%	1.57 +46.73%	1.21 +13.08%	1.09 +1.87%	1.07 +0.00%	1.07 +0.00%	1.43 +33.64%	1.40 +30.84%	1.25 +16.82%	1.26 +17.76%	1.16 +8.41%	2.30 +114.95%	2.03 +89.72%
	100% URB	1.11 +6.73%	1.10 +5.77%	n/a n/a	1.28 +23.08%	1.15 +10.58%	1.05 +0.96%	1.04 +0.00%	1.04 +0.00%	1.39 +33.65%	1.32 +26.9%	1.19 +14.42%	1.20 +15.38%	1.10 +5.77%	2.10 +101.92%	1.88 +80.77%
	100% HIG	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	2.99 +0.00%	3.21 +7.36%	n/a n/a	n/a n/a	n/a n/a	3.69 +23.41%	3.49 +16.72%
	20% SNG/URB/SUB/RUR/HIG	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	2.35 +4.44%	2.25 +0.00%	n/a n/a	n/a n/a	n/a n/a	3.17 +40.89%	2.78 +23.56%
HIGH	100% SNG	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	2.54 +12.39%	2.46 +8.85%	2.26 +0.00%	2.28 +0.88%	n/a n/a	3.38 +49.56%	3.14 +38.94%
	50% SNG/ 50% URB	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	1.91 +6.11%	1.99 +10.56%	1.81 +0.56%	1.80 +0.00%	n/a n/a	2.75 +52.78%	2.68 +48.89%
	100% URB	n/a n/a	n/a n/a	n/a n/a	1.60 +19.40%	1.40 +4.48%	n/a n/a	n/a n/a	n/a n/a	1.69 +26.12%	1.55 +15.67%	1.44 +7.46%	1.43 +6.72%	1.34 +0.00%	2.30 +71.64%	2.13 +58.96%
	100% HIG	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	5.17 +0.00%	5.42 +4.84%
	20% SNG/URB/SUB/RUR/HIG	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	n/a n/a	4.17 +9.16%	3.82 +0.00%

6.4 Impact of Battery Degradation

This section discusses the impact of battery degradation on the optimization results, which is particularly interesting for two reasons. First, the proposed optimization methodology has the unique capability of simulating and including lifetime effects in an optimization routine. Second, the implemented battery degradation model is subject to substantial uncertainty, since the battery's degradation mechanisms have not been fully understood yet. If the degradation of a battery has progressed further than was predicted at the end of the vehicle's lifetime, the power train might reach a critical system state. It is therefore of interest to understand the effects of different degradation behaviors on the optimal component design in order to determine reasonable safety margins. The following sections examine the discussed problem for two cases. The first case investigates the impact of 5 different degradation curves with similar initial degradation rates but different lifetimes. The second case examines the impact of four degradation curves with different initial degradation rates but equivalent lifetimes.

Both cases are tested for an all-purpose vehicle (20% SNG, 20% URB, 20% SUB, 20% RUR, 20% HIG) with a medium level of driver aggressiveness and an overall lifetime of 100,000km. The elimination constraint for a maximum capacity fade of 20% is removed from the optimization methodology for this study.

6.4.1 Case 1: Different lifetimes, similar degradation rates

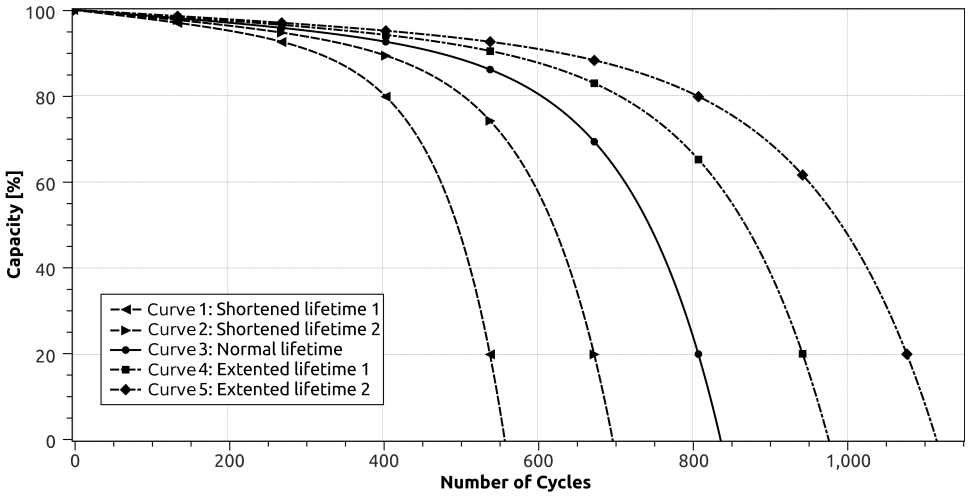


Figure 6.5. Sensitivity analysis for 5 different degradation curves with similar initial degradation rates but different lifetimes.

Table 6.5. Optimal component sizing, lifetime degradation and average fuel consumption for curves 1-5.

Design Parameter & Fitness	**Curve 1**	**Curve 2**	**Curve 3**	**Curve 4**	**Curve 5**
Max. Motor Power [kW]	39	39	39	37	37
Number of Fuel Cells	257	255	254	245	240
Battery Pack Capacity [Ah]	7.17	6.97	6.82	6.74	6.68
Avg. H_2 consumption [g/km]	2.26	2.25	2.25	2.23	2.22
Degradation at 100,000km [%]	12.0	8.2	6.1	4.6	3.8

Fig. 6.5 illustrates the 5 different degradation curves examined for case 1, and Table 6.5 provides the results of the optimization.

None of the examined batteries reached the critical capacity fade of 20% or more at the end of the vehicle's lifetime. The maximum degradation is 12% for curve 1 and the minimum degradation is 3.8% for curve 5. Hence, all batteries are still in a state with relatively low degradation rates at the end of their lifetimes. A sufficient safety margin therefore exists if the modeling uncertainties only concern the lifetime prediction of the battery.

Despite the small differences in degradation, changes in the optimal power train design are observable. The optimal battery capacity for the battery with the shortest lifetime is 7% larger than for the battery with the longest lifetime. Due to the extra weight of a larger battery pack, the motor and the fuel cell stack must be correspondingly more powerful. The difference in fuel efficiency between the worst case and the best case is 2%.

6.4.2 Case 2: Equivalent lifetimes, different degradation rates

Fig 6.6 shows the four degradation curves for case 2, and Table 6.6 lists the corresponding results. The curves range from a constant degradation rate to nearly no degradation until the end of the battery life.

Compared to case 1, the impact on the power train design is significant for some of the examined curves. The simulations for curve 1 and curve 2 considerably exceed the critical capacity fade of 20%. To ensure sufficient power supply at the end of the vehicle's lifetime, the capacity of the battery pack must be significantly larger than was initially required. For example, the initial capacity of the battery pack for curve 4 is 43% larger than the initial capacity for curve 1. The

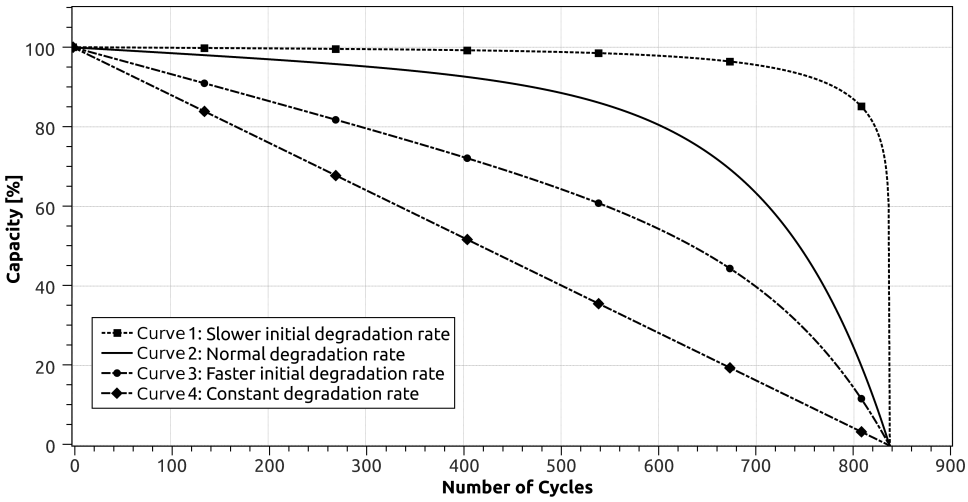


Figure 6.6. Sensitivity analysis for 4 different degradation curves with different initial degradation rates but equivalent lifetimes.

Table 6.6. Optimal component sizing, lifetime degradation and average fuel consumption for curves 1-4.

Design Parameter & Fitness	**Case 1**	**Case 2**	**Case 3**	**Case 4**
Max. Motor Power [kW]	39	39	38	40
Number of Fuel Cells	253	254	236	221
Capacity Battery Pack [Ah]	6.60	6.82	7.57	9.44
Avg. H_2 consumption [g/km]	2.23	2.25	2.28	2.45
Degradation at 100,000km [%]	0.9	6.1	25.2	42.1

optimal fuel cell stack size decreases for larger battery packs despite the increased weight of the battery. Since the battery pack acts as an energy buffer, increasing its capacity might further reduce the peak load of the fuel cell stack. Hence, smaller fuel cell stacks are sufficient to fulfill the power demand. Nevertheless, the average fuel consumption increases by up to 10% for vehicles with larger battery degradation rates due to the increased vehicle weight of resulting design.

Chapter 7

Conclusion

7.1 Summary

Large-scale optimizations of EVs have previously been too complex to address without fundamental simplification. Simplified vehicle models and optimization objectives address computational limitations but typically reduce the accuracy, versatility, and thus the reliability of the results. To overcome these limitations, it was proposed to optimize the power train design of electric vehicles on graphics processing units. The achieved computational speedup allowed for the removal of common simplifications to the model and optimization procedure. The extended, novel approach broadened the objective of the EV optimization and therefore increased the significance of the results.

Vehicle Model

To test the proposed methodology, a discrete, backward-looking model of a FCHEV was implemented in three different programming environments, which are CUDA for GPU programming, C/C++, and Matlab/Simulink. The power train of the FCHEV incorporates a PEMFC stack, Lithium-ion battery pack, motor, and EMC. The power demand to the power train is calculated by a vehicle dynamics model. During long simulation runs, a battery degradation model determines the capacity fade according to the lifetime usage of the battery. Lastly, an energy management strategy was proposed, which is easily optimized towards an application by means of only two design parameters.

Drive Cycle Generation

A novel methodology to parameterize drive cycles was presented. The proposed method is uti-

lized to stochastically analyze drive cycles or an entire driving profile. Probability functions were obtained for each parameter of a recorded set of drive cycle data. The obtained parameter results were then implemented into a drive cycle generation software tool that is capable of producing an unlimited amount of drive cycles based on the characteristics of the original data set. Due to the modular concept of the approach, it is possible to manually adjust the drive cycle parameters according to the desired objective of the optimization. It was shown that the generated drive cycles are a very good representation of the original drive cycles in terms of frequency spectra, speed distribution, acceleration distribution, load characteristics, and occurrence probabilities. In addition, the new methodology was tested with a multi-parameter design optimization of a fuel cell hybrid vehicle simulation. It was proven that the results of an optimization with generated drive cycles are almost identical to the results of an optimization with the original data set. Furthermore, the proposed methodology enhances conventional optimization methods by including entire driving profiles rather than single DCs into the objective function. This extension, along with high accuracy probability functions, accounts for uncertainties in driving patterns and for lifetime effects, such as component degradation. Moreover, concerning assurance of system stability while minimizing overall fuel consumption, the optimization results with the proposed methodology are found advantageous to conventional single-cycle optimization strategies. The stochastic approach accounts for nearly all variations of the underlying driving profile. In contrast, single-cycle optimizations are limited to a narrow selection of driving pulses. Hence, crucial driving situations may be overlooked when optimizing with just one drive cycle.

GPU-Based Optimization Methodology

In recent years, GPUs have emerged as an extremely cost-effective architecture for high performance computing. Furthermore, they have been successfully used for many applications of general purpose computing. Meta-heuristic optimization algorithms have been implemented on GPUs with significant speedups compared to sequential CPU computing. This dissertation utilized recent developments in computer science and computational hardware to advance modern optimization methodologies for engineering systems.

A two-level optimization methodology was proposed for single and multi-GPUs to determine the optimal design of EVs and FCHEVs. The methodology consists of two genetic algorithm optimization routines executed in two loops, an outer loop and an inner loop. The outer loop finds the optimal component design parameters, and the inner loop optimizes the energy management for each drive cycle. The nested framework of the proposed optimization corresponds to the CUDA pro-

gramming model, where a grid contains multiple blocks, and each block contains multiple threads. The population of the outer GA is represented by blocks and the population of the inner GA by threads. To minimize computation time, the operators of the inner GA are executed on the GPU in a fully parallel manner.

The optimization framework is generic and is therefore applicable to other vehicle architectures and different engineering systems with only minor modifications.

Performance Results

Computation times were compared for the implementation in CUDA for single and multi-GPUs, then C/C++ and Matlab/Simulink using standard CPU architectures. The CUDA code on a GPU-cluster with 8 devices performed more than 70,000 times faster than Matlab/Simulink and more than 2,000 times faster than the sequential C/C++ code. Even the slowest of the tested GPUs, a consumer-level GeForce GTX460, outperformed the C/C++ implementation by factor 164x.

It was found that the choice of population sizes has a significant influence on the convergence rate. To decrease the computation time, the population sizes should also be selected in correspondence to the CUDA programming model to maximize the occupancy of the GPUs. Futhermore, it should be considered that the nested framework of the optimization procedure exponentially increases the influence of population sizes on the convergence rate. Therefore, determining the optimal population sizes for the outer and the inner loop was concluded to be an essential procedure before the actual optimization.

Optimization results

Chapter 6 proved that lifetime optimization based on driving profiles significantly improves the quality of the obtained results. While single-cycle optimization cannot account for all possible situations, the simulation range of lifetime optimization is large enough to account for nearly all stochastic variations. Hence, the optimization results were significantly improved in terms of quality and accuracy over conventional methodologies.

It was also found that if a driver exhibits noticeable reductions in peak aggressive driving situations, the component sizes of the power train can be reduced, thus increasing the fuel efficiency by up to 9%.

Coupling the effects of differing levels of aggressiveness with various driving profiles led to optimal results for each pairing, which were then compared to reveal relative fuel efficiencies and limitations. It was found that operating a car with an optimized power train design matched for its

application and driver could lead to fuel savings of nearly 300%. However, such optimal designs may not be applicable for all applications.

Finally, an sensitivity analysis using varying parameters of the battery degradation models revealed significant effects on the optimal design. The degradation rate was the critical parameter for determining proper sizing of components. Due to its sensitive nature, understanding the effects of the degradation rate is crucial to prevent insufficiently designed components.

7.2 Contributions

Proof of Applicability

The applicability of simulating and optimizing complex engineering systems on the parallel computing architectures of GPUs was demonstrated on the example of a FCHEV.

Increase in Computation Speed

Computational speedup factors of more than 2,000x and 70,000x were achieved over a sequential C/C++ implementation and the Matlab/Simulink environment, respectively.

Improved EV Optimization Strategy

Conventional EV optimization strategies had been limited by computational constraints and therefore had been requiring simplifications of the optimization's objective and the corresponding objective function (system model). On grounds of the two previous contributions, the need for most simplifications was removed. As a consequence, key simplifications were identified and replaced by novel, more accurate methodologies. The following lists these novel approaches:

Lifetime Simulation

Instead of using just one drive cycle as input data for the objective function, the entire lifetime of the vehicle can now be simulated. The increased simulation range allows for the inclusion of the following features to the optimization procedure:

- **Application and Driver-Specific Design**
  Lifetime simulation enables accountability for all possible driving scenarios, driving patterns, driving maneuvers, and driving styles, even with small probabilities of occurrence or a chronological order assigned. It therefore allows for a greatly improved application and driver-specific design optimization.

- **Component Degradation**

  Lifetime simulations within an optimization routine allows for consideration of lifetime effects, specifically component degradation.

- **Parameter and Modeling Uncertainty**

  Most physical models of engineering systems cannot be accurately represented due to a certain degree of parameter and modeling uncertainty. The large simulation range of lifetime simulations facilitates using stochastic representations of uncertain parameters instead of average-assumed values.

Energy Management

A novel energy management control methodology for FCHEVs was proposed, which determines an efficient power split based on preliminary driving data. The energy management controller is optimized towards an application by means of only two parameters and is therefore suitable for integration into expensive optimization routines.

Two-Level Optimization Routine

A two-level optimization methodology was developed for the power train design of EVs. The proposed methodology is specifically designed for best performance on GPU architectures. The framework of the implementation is generic and can be utilized for other optimization problems with only minor modifications.

Stochastic Drive Cycle Generation

A drive cycle generation tool was developed to create input data for stochastic EV optimizations with lifetime simulations. The tool distinguishes itself from other methods by operating entirely non-deterministically in terms of data. Hence, the stochastic properties of a driving profile can either be determined from observed data or be manually adjusted according to the objective of the optimization.

7.3 Future Work

Vehicle Model

The proposed optimization methodology has been tested on a hypothetical FCHEV model. The optimization results of Section 6 are valuable findings for a better understanding of how to improve the power train design for certain applications. However, the model has not been validated and the

findings thus cannot be considered universally applicable. Therefore, the next step is to upgrade or replace the current FCHEV model with validated EV models of various propulsion designs, such as PHEVs or Battery Electric Vehicles (BEVs). Optimization results can then be used as a reference and as general guidelines for the design of modern EV power trains. The integration of a vehicle model into the optimization routine is straightforward due to the code's generic structure.

Battery Degradation

As mentioned in Section 2.3.1, the mechanisms of battery degradation, especially under dynamic loading and varying environmental conditions, are not well understood yet. Once reliable degradation models for EV applications are available, they should be incorporated into the optimization routine. The relationship between capacity fade and application and driver-specific utilization of the battery can then be examined to facilitate a better understanding of the optimal design requirements.

Optimization Algorithm

So far, only one GA (Section 4.3) has been tested with the proposed optimization methodology. During the benchmark testing of Section 5.2.1, the GA exhibited performance deviations in relation to its population size. GAs with varying population sizes [80] or different operator compositions [13] might prove to be more efficient. It is also of interest to study and compare the performance of other meta-heuristic algorithms, such as SA and ACO, when applied to the proposed optimization problem on a GPU architecture.

Other Fields of Application

The implementation framework of the proposed optimization routine is generic and can easily be adapted to solve various optimization problems on the GPU architecture. Specifically, the optimization of other engineering systems with similar fuel cell power trains, such as unmanned aerial vehicls (UAVs) or submarines, would require only minor modifications to the model. Mission profiles can then be used as stochastic input data for the objective function, which would be analogous to driving profiles for vehicles.

Considering the achieved speedup reported in Section 5.6, optimization problems, which have long been considered too complex to be solved in a reasonable time frame, might now be possible to be solved. Examples include traffic-signal optimization for entire cities or combined optimal design and control of smart grids.

Dynamic Programming

The EMC of Section 2.5 provides results, which ensure sufficient comparability of solution candidates for the FCHEV model. However, the proposed EM methodology determines a good but not necessarily optimal power-split. For other types of vehicles, such as ICE-HEVs with gear shifting, EM control is more challenging. In these cases, near-optimal solutions might be required to achieve comparability of the candidates and thus convergence of the optimization routine. Dynamic programming is an optimization technique that is capable of determining the optimal power split. Xiao et al. [97] have shown that dynamic programming is parallelizable and therefore well suited to the GPU architecture. Hence, the potential performance gain of porting dynamic programming for EM optimization to GPUs should be investigated with regard to its possible application for large-scale optimization.

Appendix A

FCHEV Model Parameter

$$v_0 = 0.279 - 8.5 \cdot 10^{-4}(T_{fc} - 298.15) + 4.308 \cdot 10^{-5} T_{fc} \left[\ln\left(\frac{p_{ca} - p_{sat}}{1.01325}\right) + \frac{1}{2}\ln\left(\frac{0.1173 \cdot (p_{ca} - p_{sat})}{1.01325}\right)\right] \tag{A.0.1}$$

$$v_a = (-1.618 \cdot 10^{-5} T_{fc} + 1.618 \cdot 10^{-2})^2 (\frac{p_{O_2}}{0.1173} + p_{sat})^2 + (1.8 \cdot 10^{-4} T_{fc} - 0.166) (\frac{p_{O_2}}{0.1173} + p_{sat}) + (-5.8 \cdot 10^{-4} T_{fc} + 0.5736) \tag{A.0.2}$$

$$c_1 = 10 \tag{A.0.3}$$

$$p_{sat} = 611 \cdot e^{0.073 \cdot (T_{fc} - 273.15) - 2.93 \cdot 10^{-4} \cdot (T_{fc} - 273.15)^2 + 9.81 \cdot (T_{fc} - 273.15)^3 - 1.9 \cdot 10^{-9} \cdot (T_{fc} - 273.15)^4} \tag{A.0.4}$$

Appendix B

Inner Algorithm

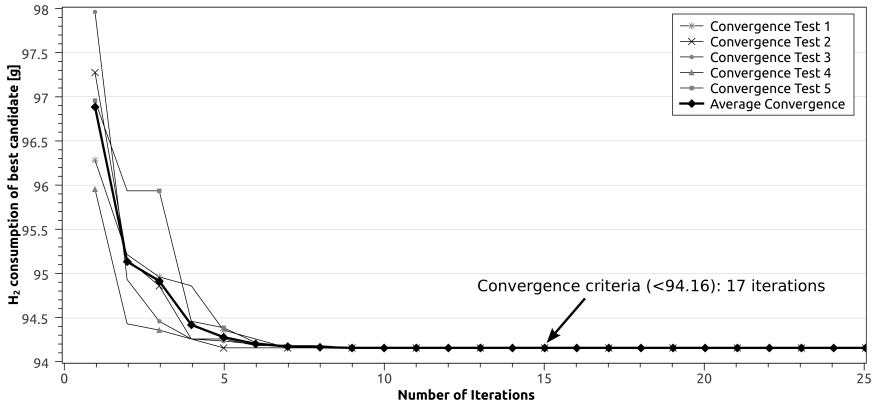


Figure B.1. Convergence study of the inner GA with population size 30.

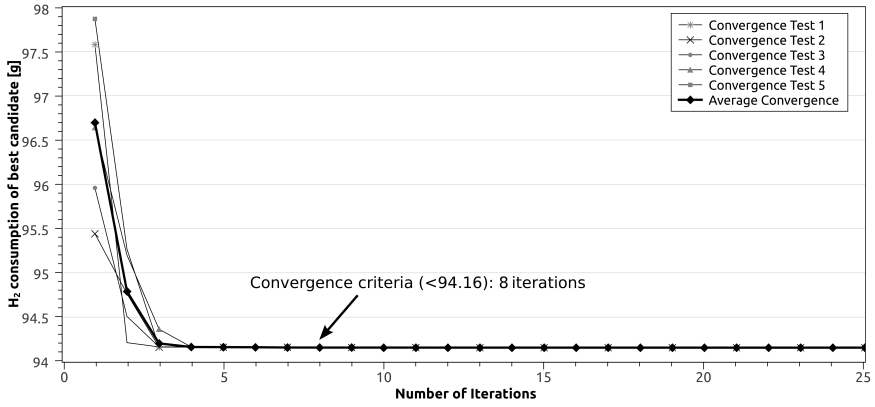


Figure B.2. Convergence study of the inner GA with population size 50.

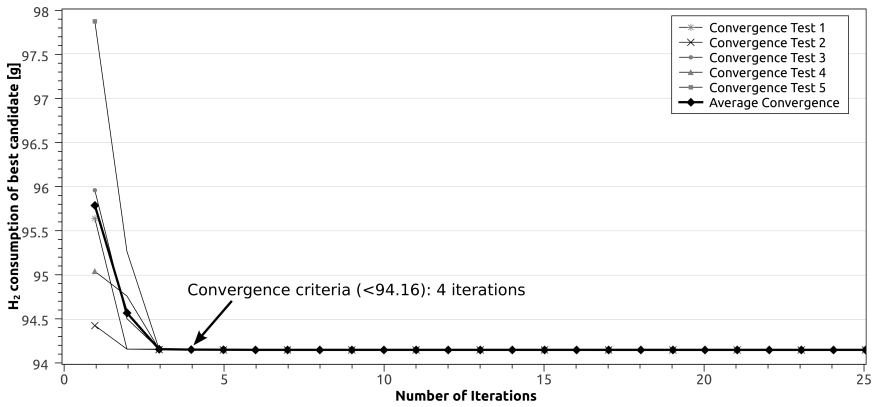


Figure B.3. Convergence study of the inner GA with population size 100.

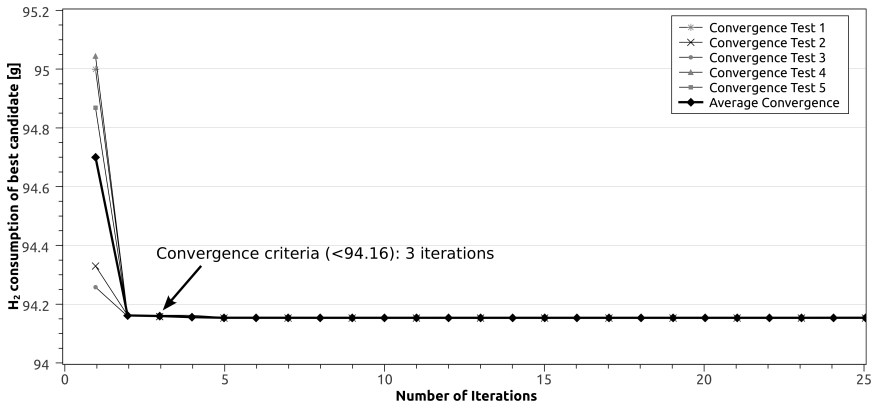


Figure B.4. Convergence study of the inner GA with population size 200.

Appendix C

Outer Algorithm

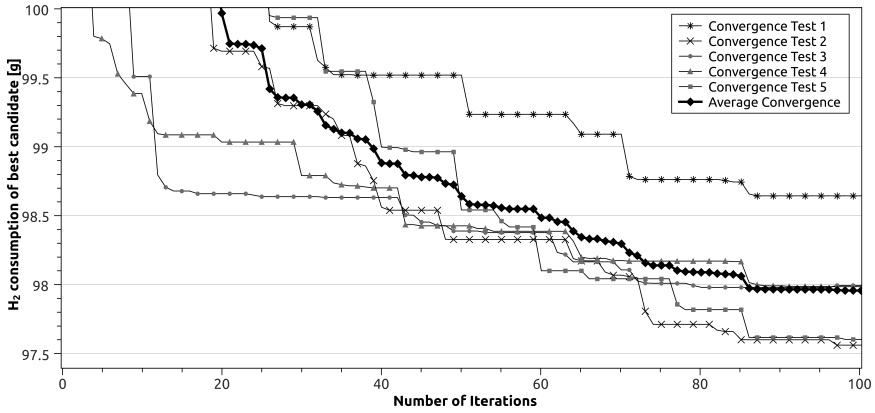


Figure C.1. Convergence study of the inner GA with population size 20.

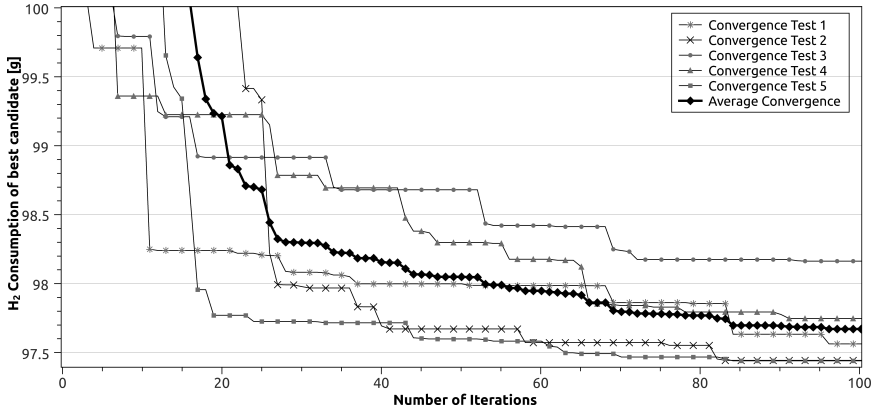


Figure C.2. Convergence study of the inner GA with population size 30.

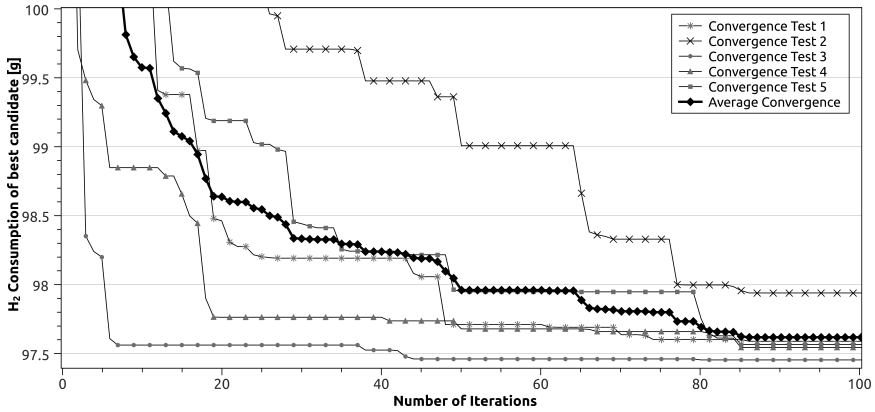


Figure C.3. Convergence study of the inner GA with population size 50.

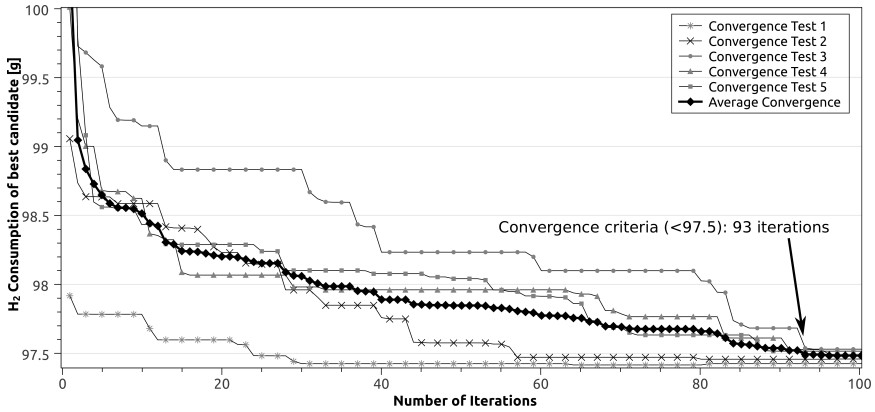


Figure C.4. Convergence study of the inner GA with population size 80.

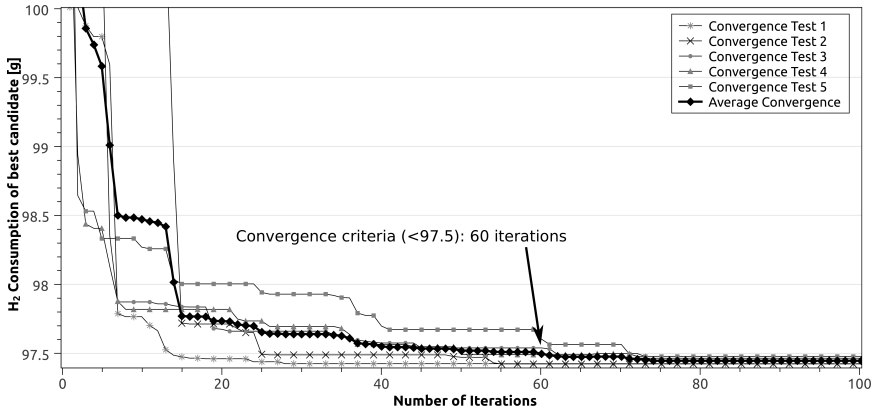


Figure C.5. Convergence study of the inner GA with population size 100.

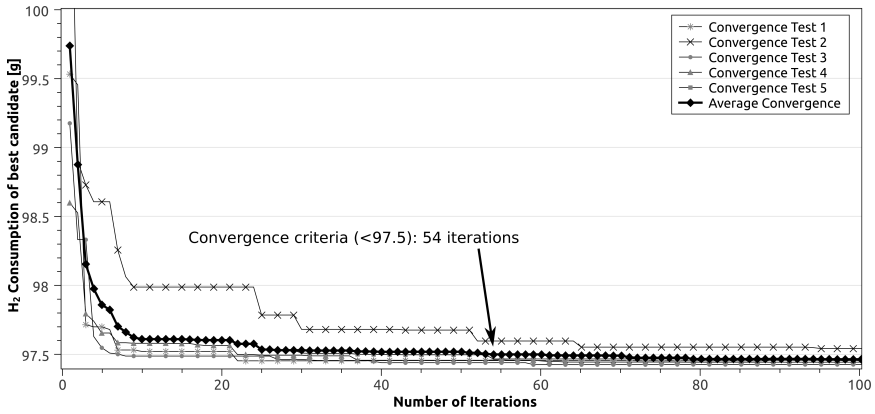


Figure C.6. Convergence study of the inner GA with population size 200.

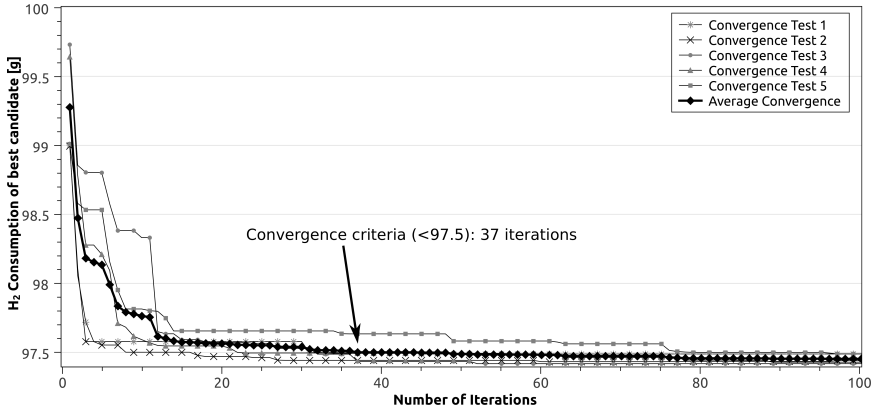


Figure C.7. Convergence study of the inner GA with population size 400.

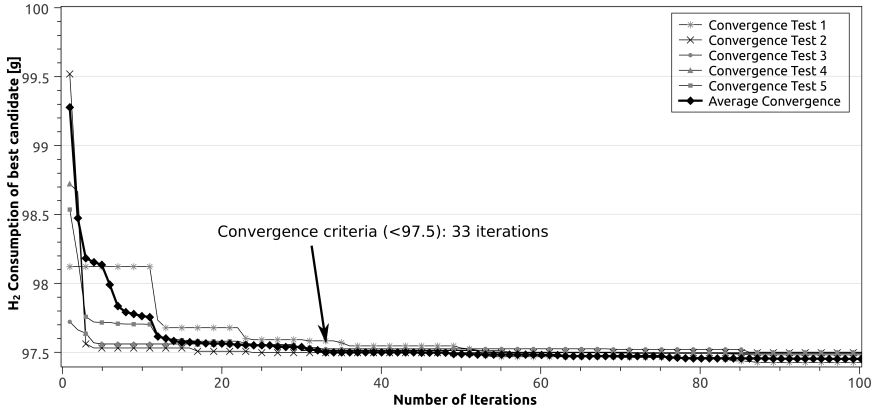


Figure C.8. Convergence study of the inner GA with population size 800.

Bibliography

[1] S.S. Rao. *Engineering optimization: theory and practice*. Wiley-IEEE, 1996.

[2] J.S. Arora. *Introduction to optimum design*. Academic Press, 2004.

[3] K. Deb. *Optimization for engineering design: algorithms and examples*. PHI Learning Pvt. Ltd., 2004.

[4] J.T. Allison. Optimal partitioning and coordination decisions in decomposition-based design optimization. *Ph.D. Dissertation, University of Michigan*, 2008.

[5] K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley and Sons, 2001.

[6] M.A. Roscher and D.U. Sauer. Dynamic electric behavior and open-circuit-voltage modeling of lifepo4-based lithium ion secondary batteries. *Journal of Power Sources*, 196(1):331–336, 2011.

[7] USABC. *Electric Vehicle Battery Test Procedures Manual, Revision 2*. Department of Energy, 1996.

[8] M.J. Mawdesley, S.H. Al-Jibouri, and H. Yang. Genetic algorithms for construction site layout in project planning. *Journal of Construction Engineering and Management*, 128(5):418–426, 2002.

[9] H. Pham. *Handbook of reliability engineering*. Birkhuser, 2003.

[10] J.T. Alander. On optimal population size of genetic algorithms. In *Proceedings of the CompEuro '92. Computer Systems and Software Engineering*, pages 65–70. IEEE, 1992.

[11] M.O. Odetayo. Optimal population size for genetic algorithms: an investigation. In *IEEE Colloquium on Genetic Algorithms for Control Systems Engineering*, pages 2/1–2/4. IEEE, 1993.

[12] A.H. Wright. *Foundations of genetic algorithms: 8th international workshop, FOGA 2005*. Springer, 2005.

[13] M. Mitchell. *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*. A Bradford Book; Third Printing edition, 1998.

[14] S.N. Sivanandam and S.N. Deepa. *Introduction to genetic algorithms*. Springer, 2007.

[15] G.E.P. Box and M.E. Muller. A note on the generation of random normal deviates. *Annals of Mathematical Statistics*, 29(2):610–611, 1958.

[16] J.G. Digakajus and K. G. Margaritis. An experimental study of benchmarking functions for genetic algorithms. *International Journal of Computer Mathematics*, 179(4):403–416, 2002.

[17] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[18] D. Kirk and W.W. Hwu. *Programming massively parallel processors: a hands-on approach*. Elsevier, 2010.

[19] V. Salmasi. Control strategies for hybrid electric vehicles: evolution, classification, comparison, and future trends. *IEEE Transactions on Vehicular Technology*, 56(5):2393–2404, 2007.

[20] H. Won and R. Langari. Fuzzy torque distribution control for a parallel hybrid vehicle. *The Journal of Knowledge Engineering*, 19(1):4–10, 2002.

[21] L. Johannesson and B. Egardt. Approximate dynamic programming applied to parallel hybrid powertrains. *Proceedings of the 17th World Congress IFAC*, pages 3373–3379, 2008.

[22] P. Rodatz, G. Paganelli, A. Sciarretta, and L. Guzzella. Optimal power management of an experimental fuel cell/supercapacitor-powered hybrid vehicle. *Journal of Control Engineering Practice*, 13(1):41–53, 2005.

[23] P. Tulpule, V. Marano, and G. Rizzoni. Energy management for plug-in hybrid electric vehicles using equivalent consumption minimisation strategy. *International Journal of Electric and Hybrid Vehicles*, 2(4):329–350, 2010.

[24] C.C. Lin, H. Peng, J.W. Grizzle, and J.M. Kang. Power management strategy for a parallel hybrid electric truck. *IEEE Transactions on Control Systems Technology*, 11(6):839–848, 2003.

[25] C.C. Lin, H. Peng, and J.W. Grizzle. A stochastic control strategy for hybrid electric vehicles. *Proceedings of the American Control Conference*, pages 4710–4715, 2004.

[26] J.T. Allison, M. Kokkolaras, and P. Y. Papalambros. On selecting single-level formulations for complex system design optimization. *ASME Journal of Mechanical Design*, 129(9):898–906, 2007.

[27] J.T. Allison, M. Kokkolaras, and P. Y. Papalambros. Optimal partitioning and coordination decisions in decomposition-based design optimization. *ASME Journal of Mechanical Design*, 131(8):709–718, 2009.

[28] R. Fellini, N. Michelena, P. Papalambros, and M. Sasena. Optimal design of automotive hybrid powertrain systems. *Proceedings of the First International Symposium on Environmentally Conscious Design andInverse Manufacturing*, pages 400–405, 1999.

[29] M.-J. Kim and H. Peng. Power management and design optimization of fuel cell/battery hybrid vehicles. *Journal of Power Sources*, 165(2):819–832, 2007.

[30] D. Assanis, G. Delagrammatikas, R. Fellini, Z. Filipi, J. Liedtke, N. Michelena, P. Papalambros, D. Reyes, D. Rosenbaum, A. Sales, and M. Sasena. An optimization approach to hybrid electric propulsion system design. *Journal of Mechanics of Structures and Machines*, 27(4):393–421, 1999.

[31] B. Zhang, Z. Chen, C. Mi, and Y.L. Murphey. Multi-objective parameter optimization of a series hybrid electric vehicle using evolutionary algorithms. *Proceedings of the IEEE Vehicle Power and Propulsion Conference*, pages 921–925, 2009.

[32] W. Gao and S.K. Porandia. Design optimization of a parallel hybrid electric powertrain. *Proceedings of the IEEE Vehicle Power and Propulsion Conference*, pages 530–535, 2005.

[33] D.W. Gao, C. Mi, and A. Emadi. Modeling and simulation of electric and hybrid vehicles. *Proceedings of the IEEE*, 95(4):729–745, April 2007.

[34] X. Liu, Y. Wu, and J. Duan. Optimal sizing of a series hybrid electric vehicle using a hybrid genetic algorithm. *Proceedings of the IEEE International Conference on Automation and Logistics*, pages 1125–1129, 2007.

[35] A. Sciarretta and L. Guzzella. Control of hybrid electric vehicles. *IEEE Control Systems Magazine*, pages 60–70, 2007.

[36] V. Schwarzer and R. Ghorbani. Stochastic optimization of an energy management controller for hybrid electric vehicles. *Proceedings of the 2010 World Electric Vehicle Symposium and Exposition (EVS25)*, pages 1–4, 2010.

[37] V. Schwarzer and R. Ghorbani. New opportunities for large-scale design optimization of electric vehicles using gpu technology. *Proceedings of the IEEE Vehicle Power and Propulsion Conference*, 2011.

[38] T.C. Austin, F.J. DiGenova, T.R. Carlson, R.W. Joy, K.A. Gianolini, and J.M. Lee. Characterization of driving patterns and emissions from light-duty vehicles in california. Technical report, Sierra Research, Inc. (Prepared for California Air Resources Board.), 1993.

[39] J. Lin and D.A. Niemeier. Estimating regional air quality vehicle emission inventories: Constructing robust driving cycles. *Journal of Transportation Science*, 37(3):330–346, 2003.

[40] E. Tazelaar, J. Bruinsma, B. Veenhuizen, and P. van den Bosch. Driving cycle characterization and generation, for design and control of fuel cell buses. *World Electric Vehicle Journal*, 3:1–8, 2009.

[41] B.Y. Liaw and M. Dubarry. From driving cycle analysis to understanding battery performance in real-life electric hybrid vehicle operation. *Journal of Power Sources*, 174:76–88, 2007.

[42] S. Shahidinejad, E. Bibeau, and S. Filizadeh. Statistical development of a duty cycle for plug-in vehicles in a north american urban setting using fleet information. *IEEE Transactions on Vehicular Technology*, 59(8):3710–3719, 2010.

[43] S. Yerramalla, A. Davari, A. Feliachi, and T. Biswas. Modeling and simulation of the dynamic behavior of a polymer electrolyte membrane fuel cell. *Journal of Power Sources*, 124:104–113, 2003.

[44] J. C. Amphlett, R. M. Baumert, R. F. Mann, B. A. Peppley, and P. R. Roberge. Performance modeling of the ballard mark iv solid polymer electrolyte fuel cell. *Journal of the Electrochemical Society*, 142(1):1–8, 1995.

[45] T. E. Springer, T. A. Zawodzinski, and S. Gottesfeld. Polymer electrolyte fuel cell model. *Journal of the Electrochemical Society*, 138(8):2234–2342, 1991.

[46] J. T. Pukrushpan, H. Peng, and A. G. Stefanopoulou. Controloriented modeling and analysis for automotive fuel cell systems. *Journal of Dynamic Systems, Measurement and Control*, 126:14–25, 2004.

[47] T. V. Nguyen and R. E. White. A water and heat management model for proton-exchange membrane fuel cells. *Journal of the Electrochemical Society*, 140(8):2178–2186, 1993.

[48] J. Larminie and A. Dick. Fuel cell systems explained. *John Wiley and Sons*, 2000.

[49] National Energy Technology Laboratory. Fuel cell handbook (seventh edition). *U.S. Department of Energy*, 2004.

[50] O. Tremblay, L. A. Dessaint, and A. I. Dekkiche. A generic battery model for the dynamic simulation of hybrid electric vehicles. *Proceedings of the IEEE Vehicle Power and Propulsion Conference*, pages 284–289, 2007.

[51] O. Tremblay and L. A. Dessaint. Experimental validation of a battery dynamic model for ev applications. *World Electric Vehicle Journal*, 3:1–10, 2009.

[52] B. Y. Liaw, G. Nagasubramanian, R. G. Jungst, and D. H. Doughty. Modeling of lithium ion cells a simple equivalent-circuit model approach. *Solid States Ionics*, 174:835–839, 2004.

[53] M. Dubarry and B. Y. Liaw. Identify capacity fading mechanism in a commercial lifepo4 cell. *Journal of Power Sources*, 194:541–549, 2009.

[54] P. Ramadass, R. White B. Haran, and B. N. Popov. Mathematical modeling of the capacity fade of li-ion cells. *Journal of Power Sources*, 123:230–240, 2003.

[55] P. Ramadass, A. Durairajan, B.S. Haran, R.E. White, and B.N. Popov. Capacity fade studies on spinel based li-ion cells. In *The Seventeenth Annual Battery Conference on Applications and Advances*. IEEE, 2002.

[56] A. Rao, G. Singhal, A. Kumar, and N. Navet. Battery model for embedded systems. In *Proceedings of the 18th International Conference on VLSI Design*. IEEE, 2005.

[57] S.J. Moura, D.S. Callaway, H.K. Fathy, and J.L. Stein. Impact of battery sizing on stochastic optimal power management in plug-in hybrid electric vehicles. In *Proceedings of the IEEE International Conference on Vehicular Electronics and Safety, ICVES*, pages 96–102. IEEE, 2008.

[58] O. Erdinc, B. Vural, and M. Uzunoglu. A wavelet-fuzzy logic based energy management strategy for a fuel cell/battery/ultra-capacitor hybrid vehicular power system. *Journal of Power Sources*, 194:369–380, 2009.

[59] O. Erdinc, B. Vural, and M. Uzunoglu. A dynamic lithium-ion battery model considering the effects of temperature and capacity fading. In *International Conference on Clean Electrical Power*, pages 383–386. IEEE, 2009.

[60] M. Chen and G.A. Rincon-Mora. Accurate electrical battery model capable of predicting runtime and iv performance. *IEEE TRANSACTIONS ON ENERGY CONVERSION*, 21(2):504–511, June 2006.

[61] A. Millner. Modeling lithium ion battery degradation in electric vehicles. In *2010 IEEE Conference on Innovative Technologies for an Efficient and Reliable Electricity Supply (CITRES)*, pages 349–356. IEEE, 2010.

[62] K. Honkura, K. Takahashi, and T. Horiba. Capacity-fading prediction of lithium-ion batteries based on discharge curves analysis. *Journal of Power Sources*, 196(23):10141 – 10147, 2011.

[63] B. Markovsky, A. Rodkin, Y.S. Cohen, O. Palchik, E. Levi, D. Aurbach, H. J. Kim, and M. Schmidt. The study of capacity fading processes of li-ion batteries: major factors that play a role. *Journal of Power Sources*, 119-121(0):504 – 510, 2003.

[64] M. Dubarry, V. Svoboda, R. Hwu, and B. Y. Liaw. Capacity and power fading mechanism identification from a commercial cell evaluation. *Journal of Power Sources*, 165(2):566 – 572, 2007.

[65] T. Osaka, S. Nakade, M. Rajamki, and T. Momma. Influence of capacity fading on commercial lithium-ion battery impedance. *Journal of Power Sources*, 119-121(0):929 – 933, 2003.

[66] E.W. Weisstein. *CRC concise encyclopedia of mathematics*. CRC Press, 2003.

[67] A. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Journal of Analytical Chemistry*, 36(8):1627–1639, 1964.

[68] J. Fallows. New life for moore's law. *The Atlantic Monthly*, 288(3):44–46, 2001.

[69] J.H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.

[70] J.H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, 2nd ed.* Cambridge, MA: MIT Press, 1992.

[71] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[72] V. Cerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[73] V. Granville, M. Krivanek, and J.P. Rasson. Simulated annealing: a proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):652–656, 1994.

[74] M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[75] D. Dorigo and T. Stuetzle. *Ant Colony Optimization*. MIT Press, 2004.

[76] G. Danzig. *Linear Programming and Extensions*. Princeton University Press, August 1998.

[77] J. Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms (Applied Optimization)*. Springer, November 2005.

[78] S. Chapra. *Numerical Methods for Engineers, 5th edition*. Mcgraw-Hill Education, August 2008.

[79] J.F. Bonnans, J.C. Gilbert, C. Lenmarechal, and C.A. Sagastizabal. *Numerical Optimization, 2nd edition*. Springer, 2006.

[80] J. Arabas, Z. Michalewicz, and J. Mulawka. Gavaps-a genetic algorithm with varying population size. In *Proceedings of the IEEE World Congress on Computational Intelligence*, volume 1, pages 73–78, 1994.

[81] www.nvidia.com. *NVIDIA CUDA C Programming Guide, Version 4.0*. 2011.

[82] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[83] D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos, and I. Buck. Gpgpu: general-purpose computation on graphics hardware. *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, 2006.

[84] M.J. Harris. *Real-Time Cloud Simulation and Rendering*. PhD thesis, University of North Carolina Technical Report TR03-040, 2003.

[85] J.C. Thibault and I. Senocak. Cuda implementation of a navier-stokes solver on multi-gpu desktop platforms for incompressible flows. *Proceedings of the 47th AIAA Aerospace Sciences Meeting Including The New Horizons Forum and Aerospace Exposition*, pages 1–15, 2009.

[86] A.D. Stivala, P.J. Stuckey, and A.I. Wirth. Fast and accurate protein substructure searching with simulated annealing and gpus. *BMC Bioinformatics*, 11(446), 2010.

[87] Dimitri Komatitsch, Gordon Erlebacher, Dominik Gddeke, and David Michea. High-order finite-element seismic wave propagation modeling with mpi on a large gpu cluster. *Journal of Computational Physics*, 229(20):7692–7714, 2010.

[88] M.Geveler, D. Ribbrock, D. Göddeke, and S. Turek. Lattice-Boltzmann simulation of the shallow-water equations with fluid-structure interaction on multi- and manycore processors. *Proceedings of Facing the Multicore Challenge*, 6310:92–104, September 2010.

[89] S. Harding and W. Banzhaf. Fast genetic programming on GPUs. In *Genetic Programming*, pages 90–101. 2007.

[90] S. Harding and W. Banzhaf. Fast genetic programming and artificial developmental systems on GPUs. In *Proceedings of the 21st International Symposium on High Performance Computing Systems and Applications*, page 2. IEEE Computer Society, 2007.

[91] K. L. Fok and T.T. Wong. Evolutionary computing on consumer graphics hardware. *Intelligent Systems, IEEE*, 22(2):69–78, 2007.

[92] S. Harding and W. Banzhaf. Distributed genetic programming on gpus using cuda. In Jos L. Risco-Martn and Oscar Garnica, editors, *WPABA'09: Proceedings of the Second International Workshop on Parallel Architectures and Bioinspired Algorithms (WPABA 2009)*, pages 1–10, 2009.

[93] P. Pospchal, J. Jaro, and J. Schwarz. Parallel genetic algorithm on the cuda architecture. In *Applications of Evolutionary Computation*, LNCS 6024, pages 442–451. Springer Verlag, 2010.

[94] Y. Sato, N. Hasegawa, and M. Sato. Gpu acceleration for sudoku solution with genetic operations. In *Proceedings of the 2011 IEEE Congress on Evolutionary Computation (CEC)*, pages 296–303, 2011.

[95] P. Pospchal, J. Schwarz, and J.Jaros. Parallel genetic algorithm solving 0/1 knapsack problem running on the gpu. In *16th International Conference on Soft Computing MENDEL 2010*, pages 64–70. Brno University of Technology, 2010.

[96] T.V. Luong, N. Melab, and E.G. Talbi. Gpu-based island model for evolutionary algorithms. In *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1089–1096, Portland, Oregon, USA, 7-11 July 2010. ACM.

[97] S. Xiao, A.M. Aji, and W.C. Feng. On the robust mapping of dynamic programming onto a graphics processing unit. In *Proceedings of the 15th international Conference on Parallel and Distributed Systems*, pages 26–33, 2009.