Large-Scale Power Electronic Circuit Simulation on a Massively Parallel Architecture

by

Shenhao Yan

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science
in
Energy Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Power electronic devices have been utilized in myriad applications in power system at all voltage and power levels. Accurate and efficient simulation is required for precise control and performance analysis of power electronic converters. Nonlinear physics-based device-level models of power electronic devices, such as insulated-gate bipolar transistor and power diode, have been previously developed to give detailed information of the components. Due to the high frequency switching transients, model complexity and nonlinear device characteristics, the application of physics-based model has hitherto been limited. Massive-thread computing brings up a new concept of dealing with highly complex models using parallel processing. Graphics processors equipped with thousands of compute cores meet the requirement of data and task parallelism for the solution of large-scale power electronic systems containing multiple detailed components while significantly increasing the simulation efficiency.

This thesis provides the application of massively parallel processing in large-scale power electronic circuit simulation. Massively parallel modules are developed for the physics-based IGBT and power diode models, in addition, efficient numerical solvers are developed for numerical linear and nonlinear system solution. The implementations are verified for the simulation of modular multi-level converters (MMCs) and compared with commercial device-level simulation software. The results show good agreement, larger-scale computational capability and considerable acceleration.

Acknowledgements

I would like to express my sincere thanks to my supervisor *Dr. Venkata Dinavahi* for his full support, encouragement, and guidance for the years throughout my research at the University of Alberta. His insightful guidance, passion and enthusiasm for the research has been an invaluable motivation for my life.

It is an honor for me to extend my gratitude to my M.Sc. committee members and for reviewing my thesis and providing invaluable comments. Special thanks go to my colleagues and friends at the RTX-Lab: *Zhiyin Zhou*, *Bo Shi*, *Ning Lin* and *Zhuoxuan Shen* .

Finally, financial help from NSERC, the University of Alberta, Government of the Province of Alberta for my living in Edmonton during these years is greatly appreciated.

Table of Contents

List of Tables

List of Figures

List of Acronyms

APIs	Application Programming Interfaces
BJT	Bipolar Junction Transistor
CUDA	Compute Unified Device Architecture
DIA	Distributed Iterative Analysis
FFT	Fast Fourier Transform
GEMES	Gaussian Elimination Matrix Equation Solution
GPGPU	General Purpose Computing on GPU
GPU	Graphic Processing Unit
HPC	High Performance Computing
HVDC	High Voltage Direct Current
IGBT	Insulated Gate Bipolar Transistor
IGBT-AE	Analog Equivalent circuit of IGBT model
IGBT-DLE	Discretized and linearized equivalent IGBT model
LPE	Linear Passive Elements
LUDMES	LU Decomposition Matrix Equation Solution
MMC	Modular multi-level converter
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
SIMD	Single Instruction Multiple Data
SM	Submodule
ODE	Ordinary Differential Equation
PCVTSM	Predictor-Corrector Variable Time-Step Method
VSC	Voltage-source Converters

1 Introduction

Computational speed is an overriding concern in large-scale power electronic circuit simulation using device-level circuit simulators. Modeling complex systems composed of power electronic subsystems such as automotive, renewable energy, and smart grid technologies, can often be very challenging as the system modeler is faced with a difficult compromise between system size, modeling complexity, and simulation duration to obtain a reasonable execution time for the application. Computational bottlenecks arise during repeated nonlinear transient simulations which are quite frequently required in cases that include but are not limited to the following:

- Robust design of power electronic systems involving optimization to fine-tune parameters at the circuit and component levels requiring several design iterations and hundreds of simulation runs.
- In addition to transient analysis, simulation-based statistical and sensitivity analysis of the system hierarchy to reduce design costs and time.
- Comprehensive fault analysis of systems using a matrix of faults representing possible device/component failures requiring multiple simulation runs to evaluate system performance and improve reliability.
- Data visualization and analysis of large sets of post-simulation results to extract meaningful system performance indices.

- Detailed modeling of complex mixed-signal and multi-domain subsystems with widely different time constants, for e.g., electronic, electrical, magnetic, thermal, and hydraulic systems.

1.1 Device-Level Model Simulation

Comparing with circuit-level and system-level simulation, device-level simulation provides possibility to observe transient response inside a single power electronic device, however, on the other hand, brings more computational complexity [1]. There are a variety of device-level simulators available for power electronic circuit simulation, both commercial and non-commercial. A partial listing of such tools include Saber®, Orcad®, PSIM®, PLECS®, LTSpice, PECS, PETS, DesignLab, etc. All of these simulation tools provide a plethora of models for semiconductor devices such as diodes, BJTs, JFETs, MOSFETs and IGBTs and fundamental circuit components such as linear/nonlinear resistors, capacitors, inductors, and independent/dependent voltage and current sources. These software tools are capable of performing an assortment of studies such as nonlinear dc, transient, linear ac (small-signal), and Monte Carlo analyses. Furthermore, since many design projects may include analog, digital, and mixed-signal simulations, most of these tools either possess native mixed-signal capabilities or they provide co-simulation interfaces to leverage the features of an external toolset [2] - [4].

Model simplifications included circuit size reduction, and using averaged or linearized models for the switching devices to observe only the system-level behavior. Model order reduction [5] was the usual course for improving computational speed in device-level circuit simulators. However, evaluating the system's comprehensive behavior entails maintaining many different models of varying size and complexity on different simulation tools. It would be far more effective if the same simulation tool could efficiently show both the system-level and device-level results over long time frames. Taking IGBT (Insulated Gate Bipolar Transistor) as an example, there are several models from detailed physics-based model to ideal switch model. Equipped with both advantages of high switching frequency of MOSFET (Metal Oxide Semiconductor Field Effect Transistor) and low conductance loss of BJT (Bipolar Junction Transistor), in addition, lower on-state resistance and higher output current capability, IGBTs have become essential devices in power converters and motor drivers. Based on modeling method, IGBT models can be classified into two categories, physics based mathematical models, which are based on semiconductor physics

and has higher accuracy, and behavioral model, which simulate its behavior using fitting algorithms and of less accuracy [10]- [12]. Hefner brought up the first complete analytical physics-based model available for circuit simulator, which is implemented in SaberRD®, consisting of a BJT with base current supplied by a MOSFET and also takes into consideration of nonlinear capacitances between terminals [7], [8]. For behavioral models, lookup tables are used in [13] to obtain the voltage controlled current source, junction capacitance and resistance in equivalent circuit. The drawback of behavioral model is a requirement of precalculated database and not suitable to predict IGBT characteristic.

As the most common voltage source converter for HVDC, modular multi-level converter (MMC) is always simulated using simplified IGBT and diode model because of its complex structure consisting of a serious of submodules (SM). Device level details in IGBT and diode are not available in these simplified models. Hefner's analytical physics-based IGBT model and Lauritzen and Ma's diode model with reverse recovery [14], [15] are adopted in this thesis to achieve detailed simulation results.

1.2 Massive Parallel Architecture

A key attribute shared by currently available simulation tools in terms of program execution is that they are single-thread programs designed to run sequentially on the CPU. Although some modifications have been made for distributed processing on multiple CPUs, such as the distributed iterative analysis (DIA) in Saber® [16], the actual execution of program code on individual CPUs is still sequential. Therefore, the attained task parallelism is coarse-grained at best. The resulting computational efficiency of device-level circuit simulators was primarily derived from an increase in CPU clock speed which until the mid 2000s could be relied upon to provide the necessary acceleration. However, computer chip manufactures no longer rely on clock speed; they have transferred to multi-core CPU and many-core GPU (Graphic Processing Unit) architecture to increase chip performance. While multiple thread concepts on CPUs such as hyperthreading were introduced early on, they were hardly taken advantage of by circuit simulators mainly due to the cumbersome task of rewriting the program code to enable multiple threads of execution.

GPUs offer a massively-parallel architecture composed of hundreds of cores grouped into streaming multiprocessors that can access both shared local memories and global system memories [17]. The attained *data parallelism* is fine-grained and is of the single instruction multiple data (SIMD) type. In other words, to take advantage of the GPU's

architecture one must re-cast the device models and the numerical algorithms into the SIMD format [19]. Mature application programming interfaces (APIs) are available for SIMD abstraction such as CUDA®, DirectCompute®, and OpenCL®. The user develops C/C++ code interlaced with special constructs and functions to access the parallel cores and distributed memories on the GPU. Furthermore, optimized numerical libraries such as CUBLAS (CUDA Basic Linear Algebra Subroutines) and CUFFT (CUDA Fast Fourier Transform) are available for linear solvers and data processing. GPU-based massively-parallel processing has been used world-wide for myriad applications such as computational fluid dynamics, life sciences, medical imaging, game physics, seismic simulations, etc., and impressive acceleration has been reported [20] - [23]. For power system computation, GPUs have been used for various applications, such as transient stability simulation [24], electromagnetic transient simulation [25], dynamic state estimation [26], [27] and power flow calculation [28].

Based on GPU hardware architecture and CUDA thread abstraction, the design of massive parallel large scale power electronics system is enabled. Unlike simulation tools at hand utilize single core or multi-core CPUs using sequential programming, massive thread programming adapts to the many core GPU architecture. The CUDA abstraction is built through high throughput streaming multiprocessors, and each multiprocessor contains many cores. Acceleration through GPU parallelism is mostly effective when dealing with many similar tasks such as adding arrays of data. The structure of large scale power electronic system meets the application of many core GPU processor. IGBTs and diodes make up SMs in MMC circuit, the construction of each SM is the same and all the SMs are in a similar situation which is suitable for accelerating. This system-level parallelism is based on the MMC circuit construction and make massive parallel programming application meaningful and approachable.

1.3 Motivation for this work

There are currently only several simulation tools dealing with physics-based detail power electronic models, and the execution time is quite long. For large-scale power electronic system, the complexity of detailed model is forced to reduce for reasonable simulation time to view the system-level results. In traditional power converters regarding switching device as ideal switch, the inner structure is neglected. The number of IGBTs and diodes are large in practical power converters, showing the detail in a single device would bring

remarkable convenience for circuit design and verification.

The main difficulty of physics-based model is to solve a large number of non-linear differential equations. Each IGBT and diode requires a discretized linearized equivalent circuit using Newton-Raphson and Gear's first or second order method. For a single IGBT and diode pair, a fixed dimension conductance matrix is developed. In a system containing numbers of IGBT-diode pairs will have a large conductance matrix relating to the pair's conductance matrix. For a full matrix, there are several ways to solve the linear matrix equation such as Gaussian elimination and LU decomposition. The system-level conductance matrix for MMC has a similar structure to block diagonal matrix with several non zero off-diagonal elements. This special matrix structure makes parallelism for solving this large size conductance matrix equation possible.

GPUs offers a way to meet the grown complexity, however the GPU is originally designed to perform graphics, converting the power electronic models to suit GPU architecture requires a lot of programming skills. The most effective way is to maximize parallelism, for both system-level and device-level. The physics-based device level model for IGBT and diode used in this thesis can both be linearized and discretized into equivalent circuits. After rearranging the sequence of calculation, one power electronic device is broken down into several units which can be processed in parallel. System-level parallelism varies according to circuit structure. In most cases, IGBTs and power diodes are used as a pair and this pair come up repeatedly, which meet the system-level parallelism. In the MMC circuit structure, the system-level parallelism is to decompose the semi block diagonal matrix mathematically. Currently, there is no work in the literature that implements large-scale device-level power electronic circuit simulation on massively parallel hardware.

In addition, to increase simulation efficiency during transient analysis, variable-time step method is adopted instead of a fixed time-step. In most power converters, most portion of the simulation time is for steady-state result, while transient time points call for more calculation. Arranging source dynamically according to task instead of fixed time point is another method to benefit simulation tools.

This thesis focuses on the GPGPU application in power electronic simulation by taking advantage of the physics structure approximation and mathematical simplification. GPU offers many core compared with CPUs, massive thread programming for power electronic system is developed using CUDA platform 5.5 [29] in C++. Optimization of code is of essential importance to make models suitable for GPU programming, such as avoiding

frequent data transformation between different registers, otherwise, CPU coding can easily surpass the speed and flexibility.

1.4 Research Objectives

To accomplish the large-scale power electronic circuit simulation on a massive parallel architecture, research objectives are listed as follows,

- The massive thread mapping for linear passive elements model such as resisters, inducers and capacitors into unified lumped Norton model.
- Discretized and linearized diode model (Diode-DLE) using P. O. Lauritzen and C. L. Ma's model [14].
- Analog equivalent circuit of IGBT using Hefner's model [8] and its conversion into a discretized and linearized equivalent IGBT model (IGBT-DLE).
- Newton-Raphson iterative method to solve nonlinear equation numerically.
- Gaussian Elimination and LU decomposition comparison for a single device and a system containing many non linear power electronics.
- Method to update the semi-block diagonal Jacobian matrix equation of MMC using a relaxation algorithm.
- Development of partial LU decomposition to solve Jacobian matrix equation and meet parallelism.
- Variable time-step scheme using the predictor and corrector method.
- Power electronic system case study of MMC circuit simulation, result evaluation for accuracy and time consumption.

1.5 Thesis Outline

This thesis contains 5 chapters. The rest of the chapters are outlined as follows:

- Chapter 2 introduces background of the GPU hardware architecture and implementation of CUDA abstraction.

- Chapter 3 describes the algorithms used to simulate a large-scale power electronic system, including the physics-based device models of linear passive elements and nonlinear power diode and IGBT, Newton-Raphson method and matrix equation solver and their massive thread parallel implementation.
- Chapter 4 presents the case study for MMC circuit of which output voltage, current and execution time are compared with SaberRD®.
- Chapter 5 gives the conclusion and some future work of the thesis.

2 Background on GPGPU

With the development of modern CPUs, the GPUs are experiencing updating among various generations. Non-graphical application of GPUs, known as GPGPU has become practical and popular. The Fermi™ generation architecture has brought not only exceptional gaming performance, but also High Performance Computing (HPC) capability. Based on evolution of Fermi™, next generation Kepler™ architecture offers higher performance with improvement of programming flexibility for multiple high level languages. In addition, a comparison between the latest generation Pascal™ and Kepler™ will be made to show the big potential of GPU development.

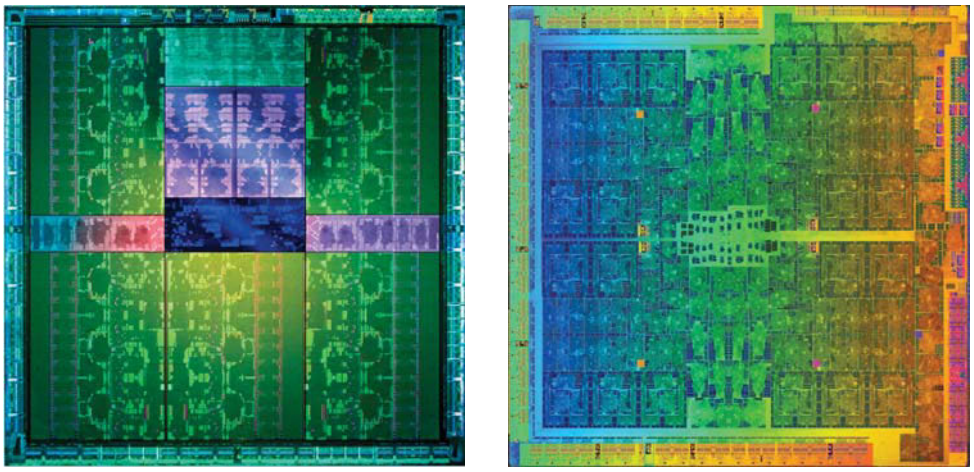
The GPU hardware architecture and CUDA abstraction will be described in this chapter, including the new features in GK110 GPU and potential power for various applications.

2.1 GPU Hardware Architecture

According to specification in Tab. 2.1, GK110 is of 28 nm fabrication with 7.1 billion transistors on a 561 mm^2 die as shown in Fig. 2.1(a), while GP104 is of higher performance and power efficiency using 16nm fabrication with 7.2 billion transistors. In hybrid computational systems, CPU and GPU cooperate as host and device respectively. Before device side execution, host sends instructions and data to device side through PCIe 3.0×16 interface at the bandwidth up to 15.754 GB/s. Instructions will be distributed though GigaThread to each streaming multiprocessor(SMX). And data from host memory will be transfered to

Table 2.1: GPU Specification

Chip	GK110	GP104
Fabrication	28 nm	16 nm
Transistors	7.1 billion	7.2 billion
Die size	561 mm^2	314 mm^2
Bus interface	PCIe 3.0 $\times$ 16	PCIe 3.0 $\times$ 16
Number of single precision cores	2880	2560
Memory bandwidth	336 GB/s	320 GB/s
Memory bus width	384 bit	256 bit
Memory size	6 GB	8 GB
Memory type	GDDR5	GDDR5X
Core clock	837 MHz	1607 MHz



(a) Die of NVIDIA® GK110 (b) Die of NVIDIA® GP104

Figure 2.1: Die of NVIDIA® GK110 and GP104

Table 2.2: Processing Power Comparison Between CPU and GPU

	Base Clock	GFLOPS(SP)	GFLOPS(DP)
GeForce GTX Titan Black (Kepler)	889M Hz	5121	1707
GeForce GTX 1080 (Pascal)	16.7M Hz	8228	257
Intel i7-3770(Ivy Bridge)	3.4G Hz	108.8	54.4

global memory on GPU board. After that, in each SMX, warp scheduler distribute every 32 threads into a warp as execution unit, which means, at most 32 threads can operate simultaneously for each warp scheduler while other threads will be parallelized with pipeline by the device automatically. After computation, result data will be saved in global memory

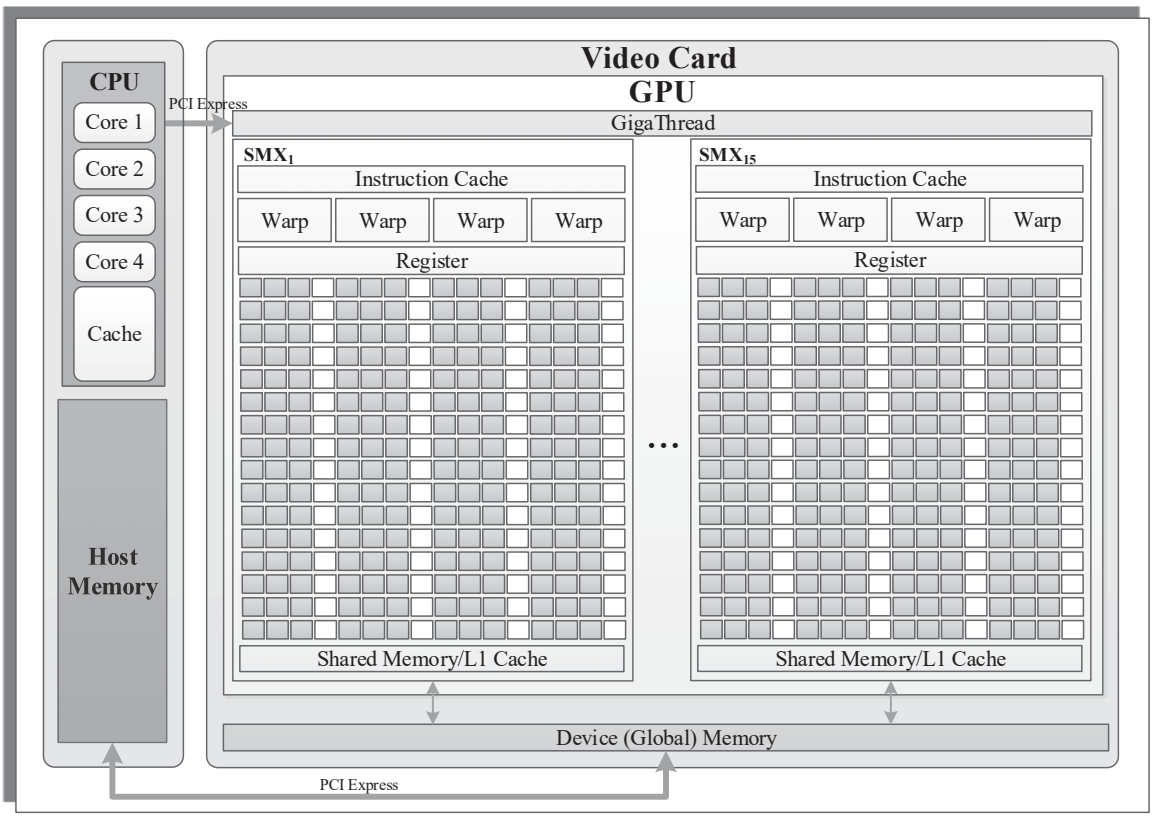


Figure 2.2: Hybrid computing system structure

and then transfer back to host side through PCIe 3.0 interface.

GK110 architecture contains 15 streaming multiprocessor (SMX) with registers, caches and shared memory in each SMX as in Fig. 2.2. Each SMX contains 192 single precision CUDA cores, which is 3 times of the 32 cores in Fermi^{TM} and 64 double precision units. A CUDA core executes one instructor of floating point or integer for a thread. In Tab.2.2, GPUs has higher computing power in both single and double precision computation than traditional CPUs. The new features in GK110 SMX is the higher double precision performance for wider computational applications.

There are several type of memories on board, including global memory, shared memory and registers. The registers inside a SMX is the fastest one but with limited number; shared memory has access to all cores inside the SMX with low latency; and global memory has a great amount with the scope to the entire device while accessed in high latency. As listed in Tab. 2.1, shared memory can be configured as the size of 16KB, 32KB or 48KB, which shares the 64KB on-chip memory with L1 Cache. The memory management, including allocation and organization, is the key of programming efficiency. One of the optimal methods is to reduce the operation with global memory and take full advantage of shared

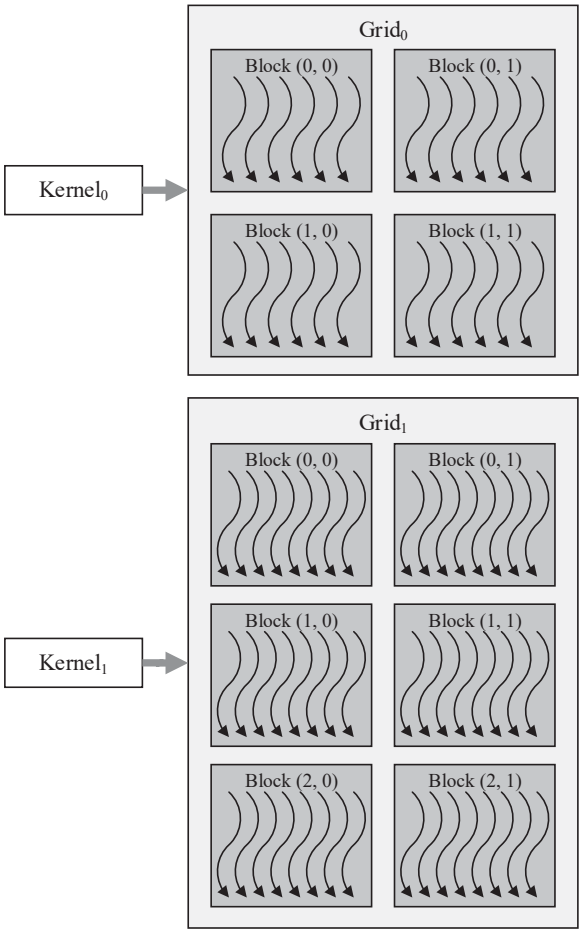


Figure 2.3: Thread hierarchy in CUDA

memory and registers according to their lifetime and scope.

2.2 CUDA Programming

CUDA offers both a platform and programming model for GUGPU programming. NVIDIA® is improving CUDA version to adapt the GPU hardware and meet the computational requirement. CUDA is designed based on the GPU hardware architecture with scalable models, which makes parallel programming on the GPU can benefit various applications. In addition, CUDA supports multiple languages, such as C/C++, Fortran and Python.

In heterogeneous programming, the CPU works as a host running serial code while GPU accomplish the parallel computing on device side.

2.2.1 Thread Management

The function involving data parallelism is referred as a *kernel*, which organizes massive *threads* into *blocks* inside a *grid*, as shown in Fig. 2.3.

The following example demonstrates the parallel matrix addition.

```
//Kernel definition
__global__ void kernel(float A[N][N], float B[N][N], float
C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.x;
    if (i < N && j < N) C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    //Kernel invocation for adding two matrix
    dim3 nthread(32, 32);
    dim3 nblock((N - 1)/nthread.x + 1, (N - 1)/nthread.y +
1);
    kernel<<<nblock, nthread>>>(A, B, C);
    ...
}
```

The key word `_global_` defines the kernel as a global function of parallel matrix addition called from host side. The kernel is configured before execution, of which threads, blocks and grid are defined according to the computing task and GPU resource. Thread is the parallel processing unit which is executed by each core, and its index can have up to three dimensions in a block. In above example, a two-dimensional block containing 32 $\times$ 32 threads is allocated to meet the two-dimensional matrix.

There is a limitation of thread numbers per block and the maximum dimension size of a grid. In compute capability 3.5 hardware, the maximum dimension size of a block is 1024$\times$1024$\times$64; the maximum dimension size of a grid is 2147483647$\times$65535$\times$65535. Additionally, the maximum thread number per block is 1024, which restricts the block structure as well. The grid, block, thread hierarchy represents the architecture of the GPU, SMX and core. Each thread in a block has an identical index *threadIdx*, which can be one-dimensional (threadIdx.x), two-dimensional (threadIdx.x, threadIdx.y) or three-dimensional (threadIdx.x, threadIdx.y, threadIdx.z), according to the block structure. Similarly, each block has a built in index *blockIdx*. As shown in Fig. 2.4, the N$\times$N matrix in the example is divided into blocks with two-dimensional block indexes (blockIdx.x, blockIdx.y). In case that the dimension N may not be evenly divided by the number of

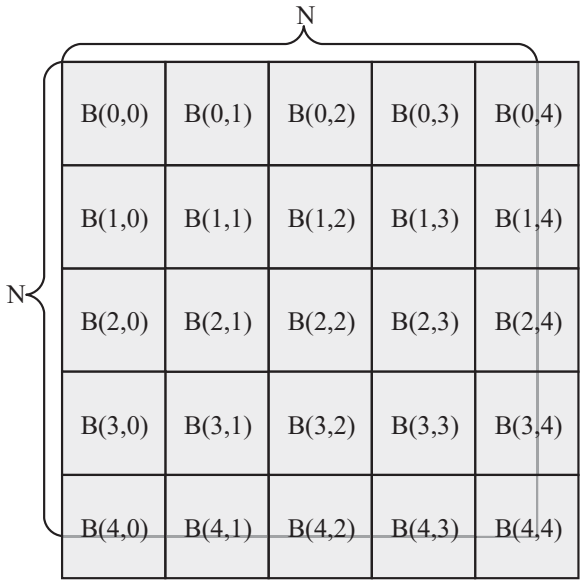


Figure 2.4: Block structure of a N× N matrix

thread in that dimension, `((N - 1)/nthread.x + 1, (N - 1)/nthread.y + 1)` is assigned to the block number per grid.

All threads inside a kernel must be synchronized before the end of execution. Block-level synchronization barrier, a synchronization point makes sure all threads inside a block has reach the command line and ready for next instructions, from the device side. Device-level synchronization barrier sets synchronization point to make sure all threads inside a grid complete all preceding requested tasks before the next kernel execution.

Another important new feature of GK110 is dynamic parallelism which enables launching nested kernels as shown in Fig. 2.5. Since the data transmission between GPU and CPU through PCIe3.0 interface is the most time consuming part in GPU programming, moving the top-level loop onto the GPU not only reduces the burden to marshal and transfer the operating data but increases the utilization of the precious computational capability of the GPU. In Fig. 2.5(a), launching multiple kernels without dynamic parallelism makes that the GPU works as an co-processor and the CPU has high occupancy for host control. In Fig. 2.5(b) Due to launching nested kernels with dynamic parallelism, the GPU can be more autonomous and allocate resource flexibly according to task load. The launching depth can reach up tp 24 generations within the limitation of GPU resources, which benefits in recursive parallel algorithms containing loops and conditions substantially.

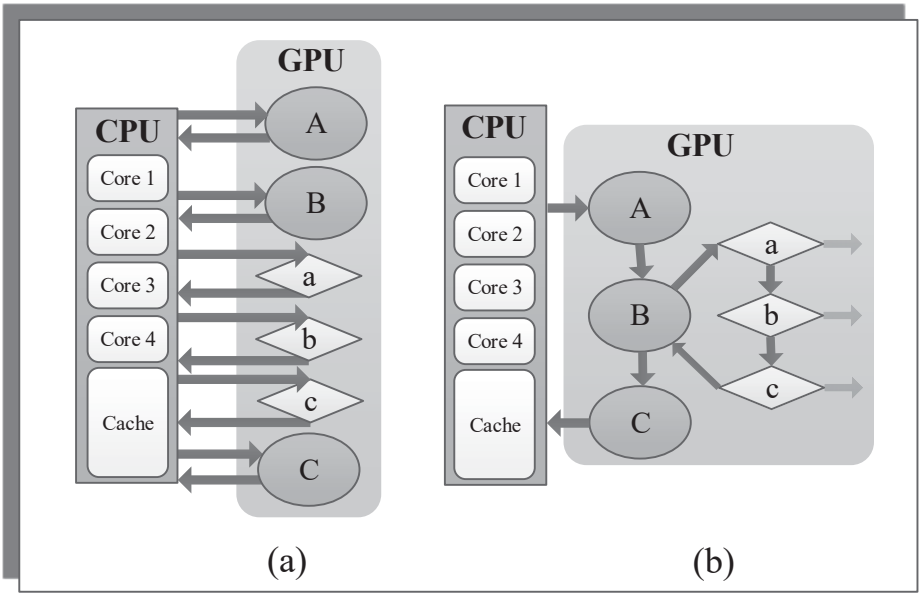


Figure 2.5: (a) Launching multiple kernels without dynamic parallelism, (b) Launching nested kernels with dynamic parallelism

2.2.2 Memory Management

As shown in Fig. 2.2, the data flow in hybrid computing system is between CPU(host) and GPU(device). The host memory is on the CPU side for data transaction with the GPU, which should be allocated before kernel execution. Based on the limitation of PCIe 3.0 interface bandwidth, the host memory should be page-locked (pinned) memory. In this way, physic memory (RAM) is directly requested by DMA on the GPU without involving pinned buffer on CPU like normal virtual memory. Global memory is off-chip memory on the GPU board making transaction with host memory and other on-chip device memory. It need to be allocated so that memory coping from host can have destination, and freed after transferring computation result back to host. Since CUDA device use unified virtual address [29], host and device memory has same pointer type so that data transaction can be conveniently achieved. Compared with shared memory, global memory has higher latency and lower bandwidth, which makes utilizing shared memory with the scope and lifetime of single block to replace the usage of global memory. In a CUDA compute capability 3.5 system, shared memory size has the flexibility of 16kB, 32kB or 48kB. The register numbers has a limitation for each block of 65536 with lowest latency and scope of thread. Therefore, the size limitation of shared memory and register numbers affect the planning of thread and block structure.

2.2.3 Error Handling and Event Management

All CUDA API calls return `cudaError_t` value indicating error types. Errors from calling a kernel, which are asynchronous errors, can't be returned before completing the kernel. This feature makes debugging code on device part difficult. The only way to check asynchronous errors is to synchronize device right after calling the kernel, and check the error code right away. Errors can be retrieved by loading the last error in the error variable. Synchronizing and checking error part can only be accomplished on the host side, which means dividing a kernel into small parts and retuning to host memory every time is a practical way for error checking.

CUDA runtime API offers functions to record events at any point of the program asynchronously. The elapsed time can be obtain after completing the event in milliseconds with a resolution of around 0.5 microseconds. It uses the timer on GPU to avoid synchronization interface.

2.3 Summary

GPU computation has a specific application capability for massive thread programming based on its hardware architecture. The Kepler™ GK110 equipped with 7080 transistors contains 15 SMX featuring 192 single-precision 64 double-precision CUDA cores each. While the new Pascal™ GP104 is the currently most efficient and fastest GPU with 2560 single-precision cores [18]. Furthermore, CUDA is abstracted based on GPU hardware architecture for massive thread programming. In large scale power electronic circuit application, the circuit characteristics and GPU programming features are both considered for efficient simulation.

3

Massive-thread Parallel Device-level Modules

Device-level power electronic circuit simulation is computationally so burdensome that engineers are forced to make model simplifications or reduce system size to obtain a reasonable execution time. This chapter proposes a massive-thread parallel simulation of large-scale power electronic circuits employing device-level modeling to obtain higher data throughput and lower execution time. Parallel massive-thread modules are proposed for the physics-based IGBT and power diode components.

3.1 Linear Passive Elements

3.1.0.1 Model Formulation

Linear passive elements (LPEs), including resistance, inductance, capacitance can be discretized by using the trapezoidal rule of integration. Each inductance and capacitance in an arbitrary combination linear system shown in Fig. 3.1(a) can be modeled as a Norton impedance network consisting of an equivalent resistance and a history current source

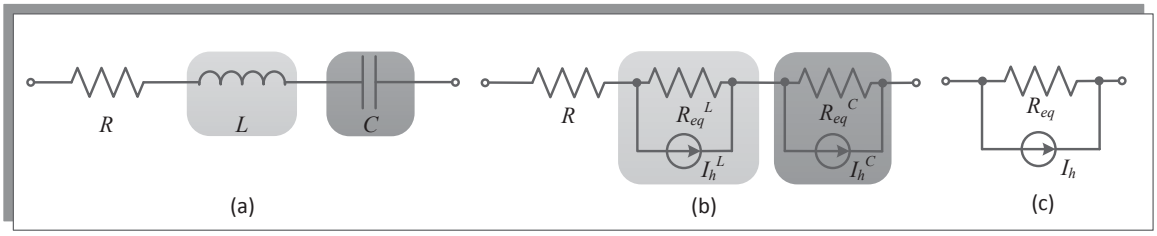


Figure 3.1: (a) An arbitrary linear passive elements combination, (b) Discretized equivalent Norton impedance of LPE, (c) Simplified Norton lumped model

shown in Fig. 3.1(b), whose parameters are given as

$$i^L(t) = \frac{v^L(t)}{R_{eq}^L} + I_h^L(t - \Delta t), \tag{3.1}$$

$$R_{eq}^L = \frac{2L}{\Delta t}, \tag{3.2}$$

$$I_h^L(t - \Delta t) = \frac{v^L(t - \Delta t)}{R_{eq}^L} + i(t - \Delta t), \tag{3.3}$$

$$i^C(t) = \frac{v^C(t)}{R_{eq}^C} + I_h^C(t - \Delta t), \tag{3.4}$$

$$R_{eq}^C = \frac{\Delta t}{2C}, \tag{3.5}$$

$$I_h^L(t - \Delta t) = -\frac{v^C(t - \Delta t)}{R_{eq}^C} - i(t - \Delta t), \tag{3.6}$$

Conversion between Norton equivalent circuits and Thevenin equivalent circuits yields a simplified Norton equivalent circuit shown in Fig. 3.1(c).

$$i(t) = \frac{v(t)}{R_{eq}} + I_h(t - \Delta t), \tag{3.7}$$

$$R_{eq} = R_{eq}^L + R_{eq}^C + R, \tag{3.8}$$

$$I_h(t - \Delta t) = \frac{I_h^L(t - \Delta t)R_{eq}^L + I_h^C(t - \Delta t)R_{eq}^C}{R_{eq}}, \tag{3.9}$$

3.1.0.2 Parallel Massive-thread Mapping

All linear elements in power systems can be transferred into an unified lumped Norton model, which makes it possible to be processed in parallel in the same kernel. And from (3.1) to (3.6), history current I_h^L and I_h^C are updated as

$$I_h^L(t) = 2i(t) - I_h^L(t - \Delta t), \tag{3.10}$$

$$I_h^C(t) = -2i(t) + I_h^C(t - \Delta t), \tag{3.11}$$

Algorithm 1 LPE Kernel

Calculate equivalent resistance R_{eq}
while $t < t_{end}$ **do**
  Calculate $i(t)$ from $v(t)$ and I_h (3.7)
  Update history current source for L (3.10)
  Update history current source for C (3.11)
  Update total history current source I_h(3.9)
  $t \leftarrow t + \Delta t$

$\triangleright\ Kernel_0$

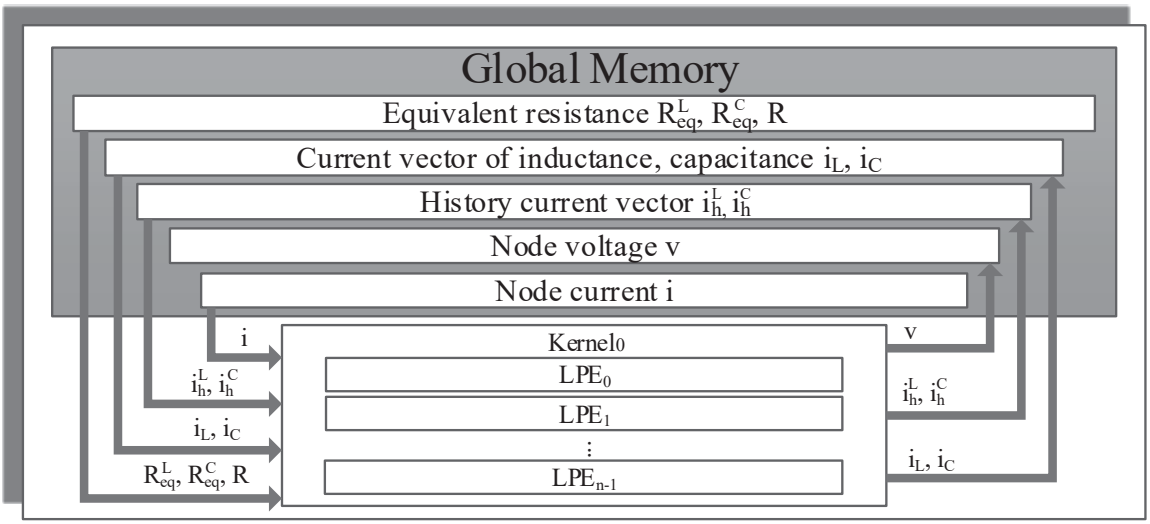


Figure 3.2: Massive thread parallel implementation of LPE elements

All the LPEs can be processed inside one kernel. There are 3 steps inside the kernel to compute at one time step. Firstly, compute unified lumped LPE current from (3.7), followed by history currents updating in each L and C form (3.10) and (3.11). Then update the LPE history current from (3.9). Each time a kernel reads and writes data from global memory in device before and after the computation inside a kernel. The massive thread parallel module for LPE is shown in Fig. 3.2.

3.2 Nonlinear Device-Level Models

3.2.1 Nonlinear Power Diode

3.2.1.1 Model Formulation

Detailed device level modeling of power diodes has a wide range of circuit operation condition since it includes equations for drift and diffusion of electrons and holes. However, different from conventional detailed model too complicated to simulate, this paper presents a simplified physics-based model containing p-i-n structure suitable for power

diode's work condition of high-voltage and fast switching. Based on Linvil's lumped charge concept [6] and derivation from semiconductor charge transport equations, this model adopts reverse recovery, junction capacitance and contact resistance to present its physics.

In a p-i-n structure diode, reverse recovery happens when turn off a forward conducting diode rapidly described as following equations:

$$i_R(t) = \frac{q_E(t) - q_M(t)}{T_M}, \tag{3.12}$$

$$0 = \frac{dq_M(t)}{dt} + \frac{q_M(t)}{\tau} - \frac{q_E(t) - q_M(t)}{T_M}, \tag{3.13}$$

$$q_E(t) = I_S\tau(e^{\frac{v_E(t)}{V_T}} - 1), \tag{3.14}$$

Where $i_R(t)$ is the diffusion current in i region, $q_E(t)$ represents charge variable in junction area, $q_M(t)$ represents charge variable in the middle of i region, T_M is the diffusion transit time across i region, τ is the lifetime of recombination, I_S is the diode saturation current constant, v_E is the junction voltage and V_T is the thermal voltage constant.

The voltage drop across i-region $v_M(t)$ is described as

$$v_M(t) = \frac{V_T T_M i(t)}{q_M(t)}. \tag{3.15}$$

Contact resistance is presented as an internal resistance R_S and has following expression,

$$v(t) = 2v_M(t) + 2v_E(t) + R_S i(t), \tag{3.16}$$

where $v(t)$ is the voltage across the diode and $i(t)$ is the total diode current. The charge of junction capacitance $q_j(t)$ contributes to a part of $i(t)$ as following,

$$i(t) = i_E(t) + \frac{dq_J}{dt}. \tag{3.17}$$

Charge $q_J(t)$ in the capacitance $C_J(t)$ is

$$q_J(t) = \int C_J(t) d(2v_E), \tag{3.18}$$

where

$$C_J(t) = \begin{cases} \frac{C_{J0}}{(1-\frac{2v_E(t)}{\phi_B})^m} & v_E < \frac{\phi_B}{4} \\ \\ \frac{m \cdot 2^{m+2} C_{J0} v_E(t)}{\phi_B} - (m-1)2^m C_{J0} & v_E \geq \frac{\phi_B}{4} \end{cases}, \tag{3.19}$$

C_{J0} is the zero-biased junction capacitance, ϕ_B is the built-in potential and m is the junction grading coefficient.

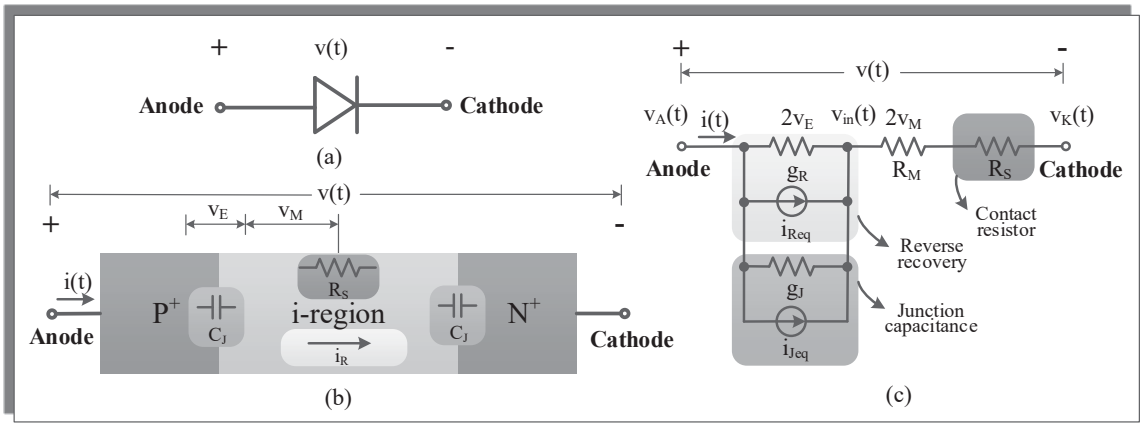


Figure 3.3: (a) Power diode symbol, (b) Physical structure of power diode, (c) Discretized and linearized equivalent circuit of power diode (Diode-DLE)

3.2.1.2 Model Discretization and Linearization

Equation (3.12) to (3.19) describe the physical based model of power diode. However, since its time-varying and nonlinear characteristic, it is necessary to obtain a discrete and linearized equivalent circuit for simulation.

Using trapezoidal rule to discretize the differential term $\frac{dq_M}{dt}$ in equation (3.13) can result following equations,

$$q_M = \frac{\Delta t \cdot q_E(t)}{2T_M(1+\frac{k_1\Delta t}{2})} + \frac{q_{hist}(t-\Delta t)}{1+\frac{k_1\Delta t}{2}}, \tag{3.20}$$

$$where\ q_{hist}(t) = \frac{\Delta t}{2T_M}q_E(t) - \frac{k_1\Delta t}{2}q_M(t), \tag{3.21}$$

$$k_1 = \frac{1}{\tau} + \frac{1}{T_M}. \tag{3.22}$$

Similarly, $\frac{dq_J}{dt}$ in (3.17) is discretized as,

$$i_J(t) = \frac{2}{\Delta t}q_J(t) - \frac{2}{\Delta t}q_J(t-\Delta t) - i_J(t-\Delta t), \tag{3.23}$$

$$where\ i_J(t) = \frac{dq_J(t)}{dt}, \tag{3.24}$$

And from (3.18) and (3.19), q_J is obtained as,

$$q_J(t) = \begin{cases} \frac{C_{J0}\phi_B}{m-1}(1-\frac{2v_E(t)}{\phi_B})^{(1-m)} & v_E < \frac{\phi_B}{4} \\ \frac{m\cdot 2^{m+2}C_{J0}v_E^2(t)}{\phi_B} - (m-1)2^{m+1}C_{J0}v_E(t) & v_E \geq \frac{\phi_B}{4} \end{cases}. \tag{3.25}$$

For nonlinear elements such as q_E in (3.14), according to (3.12) to (3.14), reverse recovery equations are linearized into an equivalent circuit consisting of a dynamic conductance

g_R and parallel current source i_{Req}. g_R is the partial derivative of $i_R(t)$ respect to $2v_E$ as following,

$$g_R = \frac{\partial i_R(t)}{\partial(2v_E(t))} = \frac{1}{2v_T} k_2 I_S \tau e^{\frac{v_E(t)}{V_T}}, \tag{3.26}$$

where,

$$k_2 = \frac{1}{T_M} - \frac{\Delta t}{2T_M^2(1+\frac{k_1\Delta t}{2})}, \tag{3.27}$$

$$\begin{aligned} i_{Req} &= i_R(t) - 2v_E(t)g_R \\ &= k_2 I_S \tau (e^{\frac{v_E(t)}{v_T}} - 1) - \frac{q_{hist(t-\Delta t)}}{T_M(1+\frac{k_1\Delta t}{2})} - 2v_E(t)g_R. \end{aligned} \tag{3.28}$$

If R_M represents equivalent resistance in terms of $2v_M$, (3.15) turns into

$$R_M = \frac{2V_T T_M}{q_M}. \tag{3.29}$$

The current i_J from junction capacitance in (3.23) is linearized as following,

$$g_J = \frac{\partial i_J(t)}{\partial(2v_E(t))} = \frac{2}{\Delta t}\frac{\partial q_J}{\partial(2v_E)} = \frac{2}{\Delta t} C_J(t), \tag{3.30}$$

$$i_{Jeq} = i_J(t) - 2v_E(t)g_J. \tag{3.31}$$

Based on the analysis above, a complete physical power diode's equivalent circuit is shown in Fig. 3.3. A 3×3 conductance matrix $\boldsymbol{G}^{Diode}$, voltage vector $\boldsymbol{V}^{Diode}$ and current vector $\boldsymbol{I}_{eq}^{Diode}$ satisfy following equation:

$$\boldsymbol{G}^{Diode} \cdot \boldsymbol{V}^{Diode} = \boldsymbol{I}_{eq}^{Diode}, \tag{3.32}$$

Where $\boldsymbol{G}^{Diode} =$

$$\begin{pmatrix} g_R + g_J & -g_R - g_J & 0 \\ -g_R - g_J & g_R + g_J + \frac{1}{R_M+R_S} & -\frac{1}{R_M+R_S} \\ 0 & -\frac{1}{R_M+R_S} & \frac{1}{R_M+R_S} \end{pmatrix}, \tag{3.33}$$

$$\boldsymbol{V}^{Diode} = [v_A \quad v_{in} \quad v_K]^T, \tag{3.34}$$

$$\boldsymbol{I}_{eq}^{Diode} = [-i_{Req} - i_{Jeq} \quad i_{Req} + i_{Jeq} \quad 0]^T \tag{3.35}$$

Applying Newton-Raphson method to obtain numerical solution for equation 3.32 yields the following equation

$$\boldsymbol{G}^{Diode(n)} \cdot \boldsymbol{\Delta V}^{Diode(n)} = \boldsymbol{\Delta I}_{eq}^{Diode(n)}, \tag{3.36}$$

where

$$\boldsymbol{\Delta V}^{Diode^{(n)}} = \boldsymbol{V}^{Diode^{(n+1)}} - \boldsymbol{V}^{Diode^{(n)}}, \tag{3.37}$$

$$\boldsymbol{\Delta I}_{eq}^{Diode^{(n)}} = \boldsymbol{G}^{Diode^{(n)}} \cdot \boldsymbol{V}^{Diode^{(n)}} - \boldsymbol{\Delta I}_{eq}^{Diode^{(n)}}. \tag{3.38}$$

To solution of (3.36) $\boldsymbol{\Delta V}^{Diode^{(n)}}$ is to update the (n+1)th iterative value $\boldsymbol{V}^{Diode^{(n+1)}}$ based on previous values, which will reach the solution of (3.32) after several iterations if converging.

3.2.1.3 Parallel Massive-thread Mapping

Algorithm 2 Diode Kernel

```
procedure PHYSICS BASED POWER DIODE MODULE
Reverse Recovery:
    Calculate q_E(t) from v_E(t) as (3.14)
    Calculate q_M(t) from q_E(t) and q_hist(t − Δt) using (3.20)
    Update q_hist(t) in (3.21)
    Calculate g_R and i_Req from v_E(t) as (3.26) and (3.28)        ▷ Kernel_0
Junction capacitance:
    Calculate C_J(t) from v_E(t) as (3.19)
    Compute g_J and i_Jeq form v_E(t) as (3.30) and (3.31)
Complete module:
    Solve matrix equation (3.32)                                    ▷ Kernel_1
    Update v_E(t)
    if v_E(t) not converged then
        go to Reverse Recovery:
    else
        Store v_A, v_in, v_K into global memory
```

A system containing n diodes, the massive-thread parallel model consists of 2 kernels as shown in Fig. 3.4. The outer loop is to solve the model using Newton-Raphson method, which will be described latter. Updating the equivalent inductance and current source from reverse recovery and junction capacitance are accomplished in Kernel$_0$. And Kernel$_1$ is to solve the matrix equation using Newton-Raphson iterative method. Then $v_E(t)$ is updated from v_A and v_{in} to compare the difference with the last iterative $v_E(t)$ to decide if converge. This loop will continue until $v_E(t)$ get converged and move to the next time point.

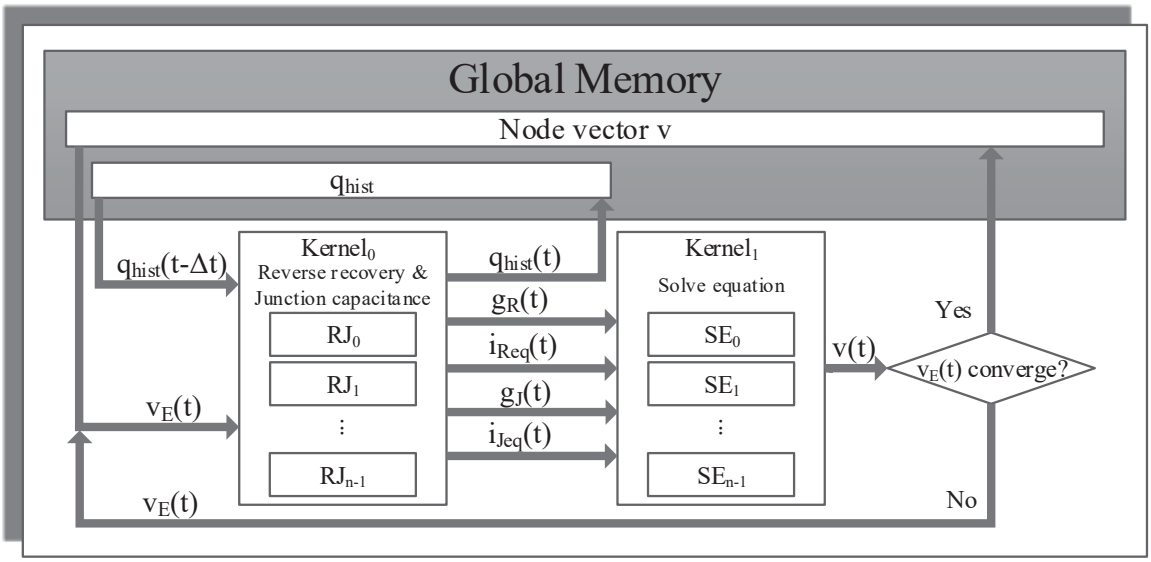


Figure 3.4: Massive thread parallel implementation of power diode

3.2.2 Nonlinear Physics-based IGBT

3.2.2.1 Module Formulation

Insulated Gate Bipolar Transistors(IGBTs) are essential in a lot of power electronic applications. Since buffer-layered IGBTs have much faster switching speed at high voltage condition, this model is developed and implemented in some commercial softwares. Hefner's physics-based model of IGBTs [8] is described to observe dynamic behavior and transient response and the results are verified by physics-based simulator SaberRD®.

Based on the model, IGBT is described as a combination of a bipolar transistor and a MOSFET. Since these internal devices are differently structured from standard microelectronic device, a regional approach is adopted to identify the phenomenological circuit of IGBT as shown in Fig. 3.5(a). An analog equivalent circuit, shown in Fig. 3.5(b), makes it possible to implement in circuit simulators by replacing the BJT with base and collector current sources and MOSFET with a current source. This analog circuit represents the currents between each of the terminal nodes and internal nodes in terms of nonlinear functions of system variables.

3.2.2.1.1 Currents The steady-state collector current i_{css} of BJT is formulated as

$$i_{css} = \frac{i_T}{1+b} + \frac{4bD_pQ}{(1+b)W^2}, \tag{3.39}$$

where b is the ambipolar mobility ratio, D_p represents hole diffusivity, the anode current i_T and quasi-neutral base width W are shown as following,

$$i_T = \frac{v_{ae}}{r_b}, \tag{3.40}$$

$$W = W_B - W_{bcj} = W_B - \sqrt{\frac{2\epsilon_{si}(v_{ds} + 0.6)}{qN_{scl}}}. \tag{3.41}$$

The instantaneous excess-carrier base charge Q is given as

$$Q = \begin{cases} Q_1 & v_{eb} \leq 0 \\ max(Q_1, Q_2) & 0 < v_{eb} < 0.6 \\ Q_2 & v_{eb} \geq 0.6 \end{cases}. \tag{3.42}$$

The emitter-base depletion charge Q_1 and excess carrier quasi-neutral base charge Q_2 has the following expression

$$Q_1 = Q_{bi} - A\sqrt{2qN_B\epsilon_{si}(0.6 - v_{eb})}, \tag{3.43}$$

$$Q_2 = p_0 qALtanh(\frac{W}{2L}), \tag{3.44}$$

where Q_{bi} is the emitter-base junction built-in charge, N_B is the base doping concentration. v_{ae} represents the voltage across r_b, q is the Electron charge, p_0 is the carrier concentration at the emitter end of the base, A is the device active area, L is the ambipolar diffusion length, W_B represents the metallurgical base width, W_{bcj} is the base-collector depletion width, ϵ_{si} is the silicon dielectric constant, v_{ds} means the drain-source voltage and N_{scl} is the collector-base space concentration. When $v_{eb} \geq 0$, p_0 can be a iterative numerical solution due to the relationship with Q_2 in (3.44) based on comparison between Q_1 and Q_2. The Newton-Raphson method is adopted to solve p_0 in the following equation

$$\frac{qv_{eb}}{kT} = ln[(\frac{p_0}{n_i^2} + \frac{1}{N_B})(N_B + p_0)] - \beta ln(\frac{p_0 + N_B}{N_B}), \tag{3.45}$$

where

$$\beta = \frac{2\mu_p}{\mu_n + \mu_p}. \tag{3.46}$$

If $0<v_{eb}<0.6$ and $Q_1>Q_2$, instead of the numerical solution, p_0 is modified as

$$p_0 = \frac{Q_1}{qALtanh(\frac{W}{2L})}. \tag{3.47}$$

The base resistance r_b in (3.40) is expressed as

$$r_b = \begin{cases} \frac{W}{q\mu_n AN_B} & v_{eb} \leq 0 \\ \frac{W}{q\mu_{eff} An_{eff}} & v_{eb} > 0 \end{cases}, \tag{3.48}$$

where μ_n and μ_{eff} stand for electron mobility and effective mobility, and n_{eff} is the effective doping concentration.

The steady-state base current i_{bss} is caused by decay of excess base charge of recombination in the base and electron injection in the emitter, expressed as following,

$$i_{bss} = \frac{Q}{\tau_{HL}} + \frac{4Q^2 N_{scl}^2 i_{sne}}{Q_B^2 n_i^2}, \tag{3.49}$$

where τ_{HL} is the base high-level lifetime, i_{sne} is the emitter electron saturation current, n_i is the intrinsic carrier concentration and Q_B represents the background mobile carrier base charge as

$$Q_B = qAWN_{scl}. \tag{3.50}$$

And the MOSFET channel current is expressed as

$$i_{mos} = \begin{cases} 0 & v_{gs} < v_T \\ K_p(v_{gs} - v_T)v_{ds} - \frac{K_p v_{ds}^2}{2} & v_{ds} \leq v_{gs} - v_T \\ \frac{K_p(v_{gs}-v_T)^2}{2} & v_{ds} > v_{gs} - v_T \end{cases}, \tag{3.51}$$

where K_p is the MOSFET transconductance parameter, v_{gs} is the gain-source voltage and v_T is the MOSFET channel threshold voltage [9].

In addition, there is a avalanche multiplication current i_{mult} in Fig. 3.5(b). Due to thermal generation in the depletion region and carrier multiplication which is an key factor to determine the avalanche breakdown voltage and the leakage current, i_{mult} is given as

$$i_{mult} = (M-1)(i_{mos} + i_{css} + i_{ccer}) + Mi_{gen}, \tag{3.52}$$

The avalanche multiplication factor M is expressed as following

$$M = [1 - \frac{v_{ds}}{BV_{cb0}}]^{-BV_n}, \tag{3.53}$$

where BV_{cb0} is the open emitter collector-base breakdown voltage and the collector-base thermally generated current I_{gen} has the following expression

$$i_{gen} = \frac{qn_iA}{\tau_{HL}}\sqrt{\frac{2\epsilon_{si}v_{bc}}{qN_{scl}}}. \tag{3.54}$$

3.2.2.1.2 Capacitance and charges The gate-source capacitance C_{gs} in analog module is a constant while all the others are charge related.

$$Q_{gs} = C_{gs}v_{gs}. \tag{3.55}$$

The gate-drain capacitance C_{gd} is as

$$C_{gd} = \begin{cases} C_{oxd} & v_{ds} \leq v_{gs} - v_{Td} \\ \frac{C_{gdj}C_{oxd}}{C_{gdj}+C_{oxd}} & v_{ds} > v_{gs} - v_{Td} \end{cases}, \tag{3.56}$$

where v_{Td} is the gate-drain overlap depletion threshold voltage, C_{oxd} is the gate-drain. And the gate-drain overlap depletion capacitance C_{gdj} is given as

$$C_{gdj} = \frac{A_{gd}\epsilon_{si}}{W_{gdj}}, \tag{3.57}$$

where A_{gd} is the gate-drain overlap area, ϵ_{si} is the silicon dielectric constant and W_{gdj} is the gate-drain overlap depletion width given as following,

$$W_{gdj} = \sqrt{\frac{2\epsilon_{si}(v_{dg} + V_{Td})}{qN_{scl}}}. \tag{3.58}$$

And the charge of C_{gd} has the expression as

$$Q_{gd} = \begin{cases} C_{oxd}v_{dg} & v_{ds} \leq v_{gs} - v_{Td} \\ \frac{qN_B\epsilon_{si}A_{gd}^2}{C_{oxd}}[\frac{C_{oxd}W_{gdj}}{\epsilon_{si}A_{gd}} - ln(1 + \frac{C_{oxd}W_{gdj}}{\epsilon_{si}A_{gd}})] - C_{oxd}v_{Td} & v_{ds} > v_{gs} - v_{Td} \end{cases}. \tag{3.59}$$

Similarly, other capacitances are related to a certain depletion capacitance and the depletion capacitance depends on active area and width. The drain-source depletion capacitance C_{dsj}, is related to (A-A_{gd}) and drain-source depletion width W_{dsj} as following,

$$C_{dsj} = \frac{(A - A_{gd})\epsilon_{si}}{W_{dsj}}. \tag{3.60}$$

And the charge Q_{ds} is as

$$Q_{ds} = A_{ds}\sqrt{2\epsilon_{si}(v_{ds} + 0.6)qN_{scl}}. \tag{3.61}$$

The emitter-base capacitance C_{eb} is solved for $\frac{\partial Q_{eb}}{\partial V_{eb}}$ according to the following equation

$$V_{ebj} = 0.6 - \frac{(Q - Q_{bi})^2}{2qN_B\epsilon_{si}A^2}, \tag{3.62}$$

as

$$C_{eb} = -\frac{qN_B\epsilon_{si}A^2}{Q - Q_{bi}}. \tag{3.63}$$

And Q_{eb} is the same as Q given in (3.42). The collector-emitter redistribution capacitance C_{cer} is solved from ambipolar diffusion equation as

$$C_{cer} = \frac{QC_{bcj}}{3Q_B}, \tag{3.64}$$

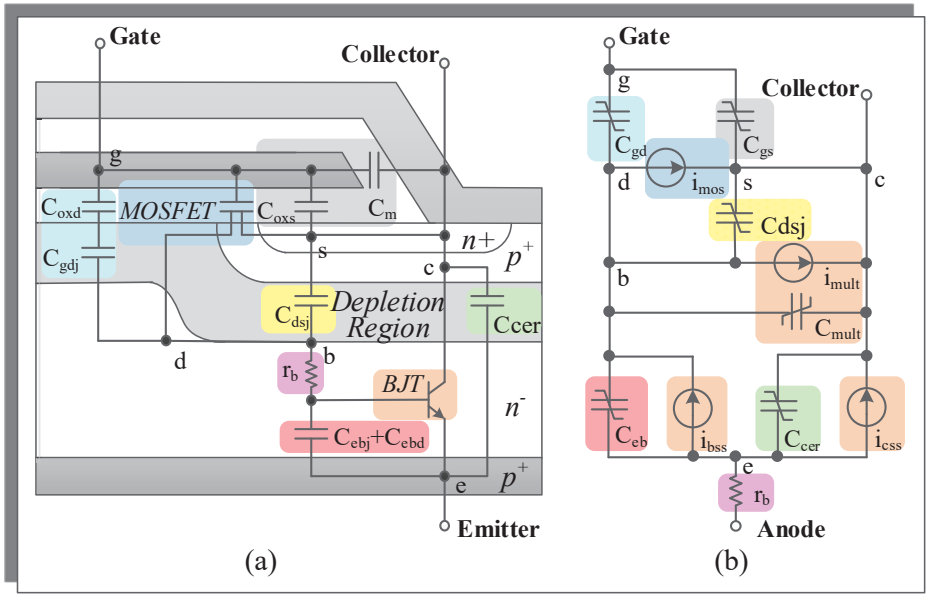


Figure 3.5: (a) Phenomenological physical structure of IGBT, (b) Analog equivalent circuit of IGBT (IGBT-AE)

where Q_B is the background mobile carrier base charge and the base-collector depletion capacitance C_{bcj} is the same as C_{dsj}. The carrier multiplication charges and capacitance are related to C_{cer} as given

$$Q_{mult} = (M-1)Q_{ce}, \tag{3.65}$$

$$C_{mult} = (M-1)C_{cer}. \tag{3.66}$$

3.2.2.2 Model Discretization and Linearization

Similar to power diode model, the analog equivalent circuit model (IGBT-AE) in Fig. 3.5(b) containing nonlinear and time variable element is transferred into discretized and linearized equivalent equivalent circuits (IGBT-DLE) as shown in Fig. 3.6.

After applying Newton-Raphson method of linearizion on four current sources and the conductivity-modulated base resistance r_b, these five elements are converted into discretized elements. Given the i_{mos} at the $(n+1)^{th}$ iteration as an example yields

$$\begin{aligned} i_{mos}^{n+1} &= i_{mos}^{n} + \frac{\partial i_{mos}^{n}}{\partial v_{gs}}(v_{gs}^{n+1} - v_{gs}^{n}) + \frac{\partial i_{mos}^{n}}{\partial v_{ds}}(v_{ds}^{n+1} - v_{ds}^{n}) \\ &= i_{moseq}^{n} + g_{mosgs}^{n} v_{gs}^{n+1} + g_{mosds}^{n} v_{ds}^{n+1}, \end{aligned} \tag{3.67}$$

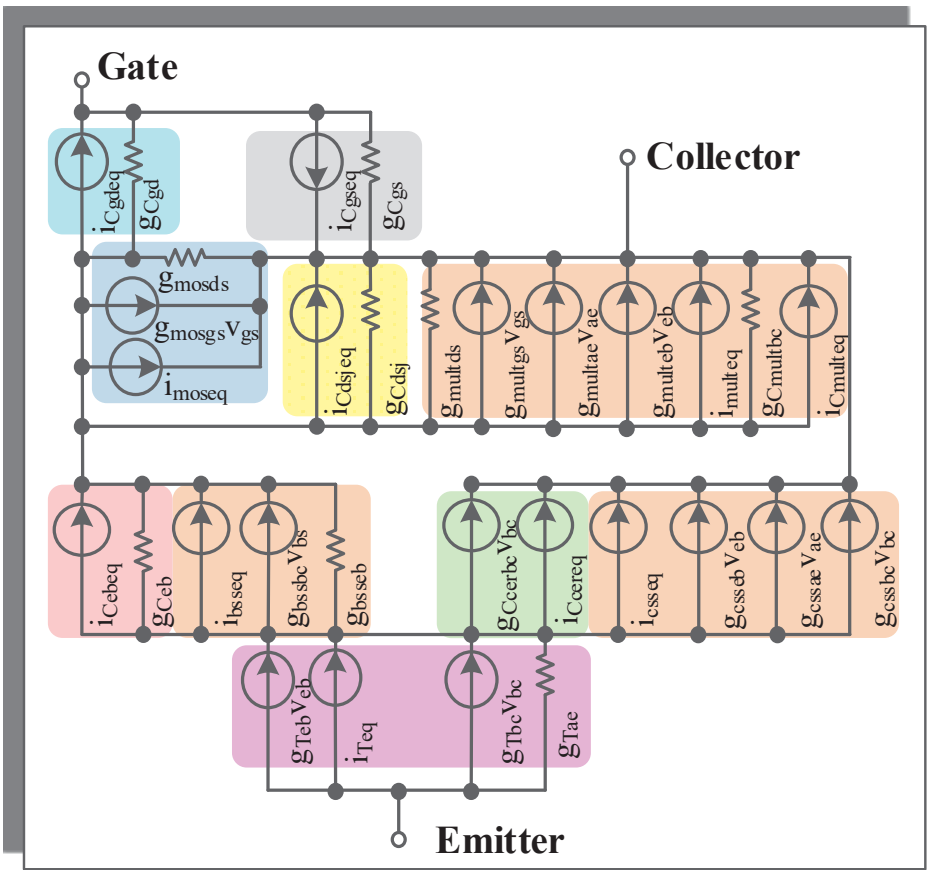


Figure 3.6: Discretized and linearized equivalent circuit of IGBT (IGBT-DLE)

where

$$i^n_{moseq} = i^n_{mos} - g^n_{mosgs} v_{gs} - g^n_{mosds} v_{ds}. \tag{3.68}$$

Similarly, applying to i_T, i_{css}, i_{bss}, i_{mult} gets equation (3.69) to (3.72) as following,

$$i^{n+1}_T = i^n_{Teq} + g^n_{Tae} v^{n+1}_{ae} + g^n_{Tbc} v^{n+1}_{bc} + g^n_{Teb} v^{n+1}_{eb}, \tag{3.69}$$

$$i^{n+1}_{css} = i^n_{csseq} + g^n_{cssbc} v^{n+1}_{bc} + g^n_{cssae} v^{n+1}_{ae} + g^n_{csseb} v^{n+1}_{eb}, \tag{3.70}$$

$$i^{n+1}_{bss} = i^n_{bsseq} + i^n_{bsseb} v^{n+1}_{eb} + g^n_{bssbc} v^{n+1}_{bc}, \tag{3.71}$$

$$i^{n+1}_{mult} = i^n_{multeq} + g^n_{multds} v^{n+1}_{ds} + g^n_{multds} v^{n+1}_{ds} + g^n_{multae} v^{n+1}_{ae} + g^n_{multeb} v^{n+1}_{eb} \tag{3.72}$$

All the capacitances are displaced with pairs of a conductance in parallel with a current source. Euler, trapezoidal or other Gear 2nd order method on charges at the t+Δt time point make it discretized. Given trapezoidal method on Q_{gs} as an example,

$$Q_{gs}(t + \Delta t) = Q_{gs}(t) + \frac{h}{2}[i_{Qgs}(t + \Delta t) + i_{Qgs}(t)], \tag{3.73}$$

solving for the i_{Qgs} yields

$$i_{Qgs}(t+\Delta t) = \frac{2}{h}[Q_{gs}(t+\Delta t) - Q_{gs}(t)] - i_{Qgs}(t) = i_{Qgseq}(t) + G_{Cgs}(t+\Delta t)v_{gs}(t+\Delta t) \tag{3.74}$$

$$i_{Qgseq}(t) = i_{Qgs}(t) - G_{Cgs}(t)v_{gs}(t). \tag{3.75}$$

With the conversion above, IGBT-DLE is modeled as in Fig. 3.6. Applying KCL to nodes Gate, Collector, Base, Emitter and Anode gives the five equations as following,

$$\frac{dQ_{gs}}{dt} - \frac{dQ_{ds}}{dt} = i_G \tag{3.76}$$

$$-i_{mult} - \frac{dQ_{mult}}{dt} - \frac{dQ_{gs}}{dt} - i_{mos} - \frac{dQ_{Ccer}}{dt} - i_{css} - \frac{dQ_{dsj}}{dt} = i_C \tag{3.77}$$

$$\frac{dQ_{gd}}{dt} + i_{mos} + \frac{dQ_{dsj}}{dt} + i_{mult} + \frac{dQ_{mult}}{dt} - i_{bss} - \frac{dQ_{eb}}{dt} = 0 \tag{3.78}$$

$$\frac{dQ_{eb}}{dt} + i_{bss} + \frac{dQ_{Ccer}}{dt} + i_{css} - i_T = 0 \tag{3.79}$$

$$i_T = i_A \tag{3.80}$$

After applying the detailed linearized equivalent circuit, a 5×5 conductance matrix $\boldsymbol{G}^{IGBT}$ is obtained to satisfy the following equation

$$\boldsymbol{G}^{IGBT} \cdot \boldsymbol{V}^{IGBT} = \boldsymbol{I}_{eq}^{IGBT}, \tag{3.81}$$

Where

$$\boldsymbol{V}^{IGBT} = [v_c \quad v_g \quad v_a \quad v_d \quad v_e]^T, \tag{3.82}$$

$$\boldsymbol{I}_{eq}^{IGBT} = [i_{ceq} \quad i_{geq} \quad i_{aeq} \quad i_{deq} \quad i_{eeq}]^T, \tag{3.83}$$

Where

$$i_{ceq} = i_{multeq} + i_{Cmulteq} + i_{csseq} + i_{Ccereq} + i_{Cgseq} + i_{Cdsjeq} + i_{moseq}, \tag{3.84}$$

$$i_{geq} = -i_{Cgseq} + i_{Cdgeq}, \tag{3.85}$$

$$i_{aeq} = -i_{Teq}, \tag{3.86}$$

$$i_{deq} = i_{Ccebeq} + i_{bsseq} - i_{moseq} - i_{Cdgeq} - i_{Cdsjeq} - i_{multeq} - i_{Cmulteq}, \tag{3.87}$$

$$i_{eeq} = -i_{csseq} - i_{bsseq} - i_{Ccereq} - i_{Ccebeq} + i_{Teq}, \tag{3.88}$$

$$\boldsymbol{G}^{IGBT} =$$

$$\begin{pmatrix} g_{multds}+g_{multgs}+g_{Cmultbc}+ g_{cssbc}+g_{Ccerbc}+g_{Cgs}+ g_{Cdsj}+g_{mosds}+g_{mosgs} & -g_{multgs}-g_{Cgs}-g_{mosgs} & -g_{multae}-g_{cssae} & -g_{multds}-g_{Cmultbc}-g_{cssbc}+ g_{csseb}-g_{Ccerbc}-g_{dsj}- g_{mosds}+g_{multeb} & -g_{multae}-g_{cssae}+ g_{csseb}-g_{multeb} \\ -g_{Cgs} & g_{Cgs}+g_{Cdg} & 0 & -g_{Cdg} & 0 \\ -g_{Tbc} & 0 & g_{Tae} & g_{Tbc}-g_{Teb} & -g_{Tae}+g_{Teb} \\ g_{bssbc}-g_{mosgs}-g_{mosds}- g_{Cdsj}-g_{multds}-g_{multgs}- g_{Cmultbc} & g_{mosgs}-g_{Cdg}+g_{multgs} & g_{multae} & g_{Ceb}-g_{bssbc}+g_{bsseb}+ g_{mosds}+g_{Cdg}+g_{Cdsj}+ g_{multds}-g_{multeb}-g_{Cmultbc} & -g_{Ceb}+g_{bsseb}- g_{multae}+g_{multeb} \\ -g_{cssbc}-g_{Ccerbc}- g_{bssbc}+g_{Tbc} & 0 & g_{cssae}-g_{Tae} & g_{cssbc}-g_{csseb}+g_{Ccerbc}- g_{bsseb}+g_{bssbc}-g_{Ceb}- g_{Tbc}+g_{Teb} & -g_{cssae}+g_{csseb}+g_{bsseb}+ +g_{Ceb}+g_{Tae}-g_{Teb} \end{pmatrix}, \tag{3.89}$$

Similarly, when using Newtone-Raphson method to solve the matrix equation (3.81), instead of solve node vector $\boldsymbol{V}^{IGBT}$ directly, $\boldsymbol{\Delta V}$ is obtained to update $\boldsymbol{V}^{IGBT}$ iteratively. Therefore, the iterative equation is given as

$$G^{IGBT}\Delta V = -I. \tag{3.90}$$

Where

$$-I = [i^c \quad i^g \quad i^a \quad i^d \quad i^e]^T, \tag{3.91}$$

$$\begin{aligned} i^c &= i_{multeq} + i_{Cmulteq} + i_{csseq} + i_{Ccereq} + i_{Cgseq} + i_{Cdsjeq} + i_{moseq} + (g_{multgs} - g_{Cgs} \\ &\quad + g_{mosgs})(v_g - v_c) + (g_{Cmultbc} + g_{multds} + g_{Cdsj} + g_{mosds} + g_{cssbc} + g_{Ccerbc})(v_d - v_c) \\ &\quad + (g_{multae} + g_{cssae})(v_a - v_e) + (g_{multeb} + g_{csseb})(v_e - v_d), \end{aligned} \tag{3.92}$$

$$i^g = -i_{Cgseq} + i_{Cdgeq} + g_{Cgs}(v_c - v_g) + g_{Cdg}(v_d - v_g), \tag{3.93}$$

$$i^a = -i_{Teq} + g_{Tae}(v_a - v_e) + g_{Tbc}(v_b - v_c) + g_{Teb}(v_e - v_b), \tag{3.94}$$

$$\begin{aligned} i^d &= i_{Ccebeq} + i_{bsseq} - i_{moseq} - i_{Cdgeq} - i_{Cdsjeq} - i_{multeq} - i_{Cmulteq} + (g_{bssbc} - g_{multds} \\ &\quad - g_{Cmultbc} - g_{Cdsj} - g_{mosds})(v_b - v_c) + (g_{Ceb} + g_{bsseb} - g_{multeb})(v_e - v_b) - g_{Cgd}(v_b - v_g) \\ &\quad - (g_{mosgs} + g_{multgs})(v_g - v_c) - g_{multae}(v_a - v_e), \end{aligned} \tag{3.95}$$

$$\begin{aligned} i^e &= -i_{csseq} - i_{bsseq} - i_{Ccereq} - i_{Ccebeq} + i_{Teq} + (g_{Tbc} - g_{cssbc} - g_{bssbc} - g_{Ccerbc})(v_b - v_c) \\ &\quad + (g_{Tae} - g_{cssae})(v_a - v_e) + (g_{Teb} - g_{bsseb} - g_{Ceb} - g_{csseb})(v_e - v_b), \end{aligned} \tag{3.96}$$

3.2.2.3 Parallel Massive-thread Mapping

For a system containing n IGBTs, the massive thread parallel implementation is shown in Fig. 3.7 and described in the following algorithm. There are 12 kernels involved in the module. Kernel_0 and Kernel_1 is to check PN junction and FET junction voltage limitation within successive Newtone-Raphson iterations. And the main equivalent parameters updating is accomplished in Kernel_2 to Kernel_9, including r_b, 6 nonlinear capacitors and 5

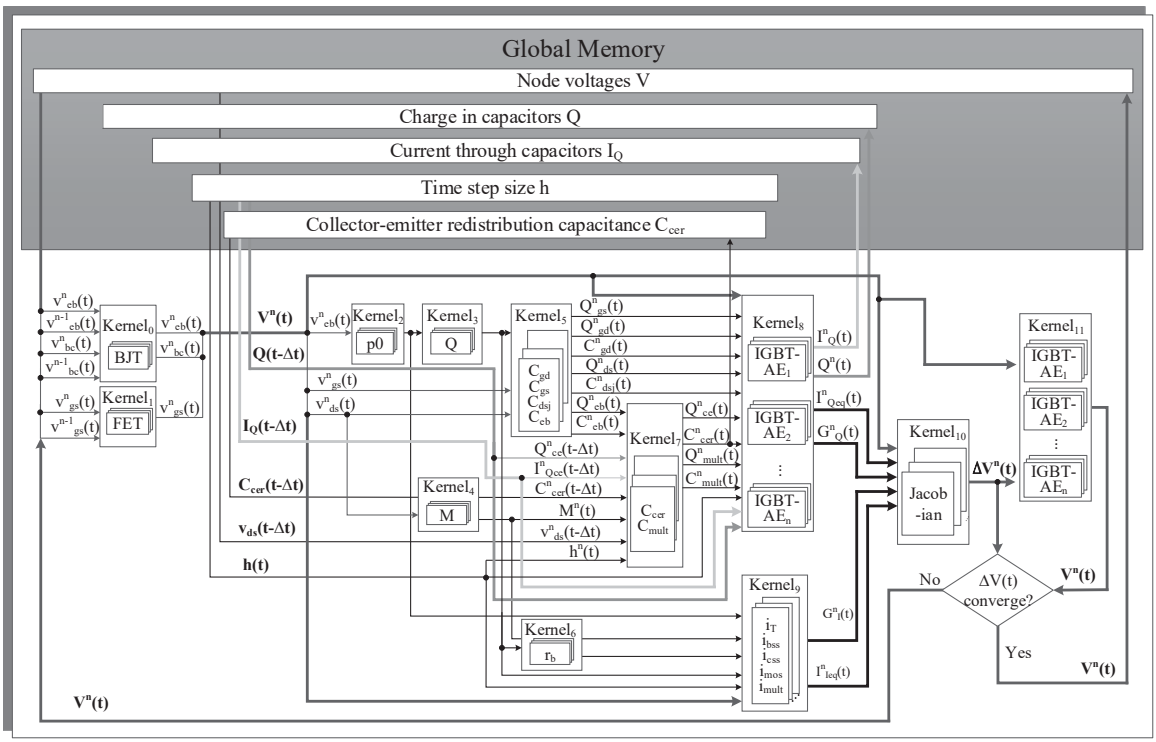


Figure 3.7: Massive thread parallel implementation of IGBT

current source approximation. And Kernel$_{10}$ is to build the iterative Jacobian matrix equation in (3.90) and solve $\boldsymbol{\Delta V^{IGBT}}$ use LU decomposition or Gaussian elimination, which will be described in detail in later sections. Kernel$_{11}$ assigns n cuda blocks containing 5 threads per block to update voltage vector in each IGBT. If $\boldsymbol{\Delta V}^{IGBT}$ converges, it means the iterative solution for current time point is accomplished and node voltage $\boldsymbol{V}^{IGBT}$ will be stored into global memory and used as the initial value for next time point. Otherwise, repeat from checking junction limitation for the next time point iterative value.

The 6 nonlinear capacitors C_{gd}, C_{gs}, C_{dsj}, C_{mult}, C_{eb} and C_{cer} are updated Kernel$_3$ to Kernel$_6$ and Kernel$_8$ for charges, currents, equivalent conductance and parallel current sources. Intermediate parameters p_0, Q, Q_1, Q_2 and M are processed in p_0, Q and M unit in Kernel$_3$, Kernel$_4$ and Kernel$_5$. Capacitance and charges in these 6 capacitors are updated in Kernel$_6$ and Kernel$_8$. Followed by equivalent conductance $\boldsymbol{G}_Q$ and parallel current source $\boldsymbol{I}_{Qeq}$ update in Kernel$_{10}$. Similarly, for the approximation to current sources i_{mos}, i_T, i_{css}, i_{bss} and i_{mult}, Kernel$_{10}$ updates $\boldsymbol{G}_I$ and $\boldsymbol{I}_{Ieq}$. In this way, parameters in IGBT-DLE are updated. Solving for $\boldsymbol{\Delta V}^{IGBT}$ in the matrix equation (3.90) is accomplished in Kernel$_{11}$. And Kernel$_{12}$ is to update $\boldsymbol{V}^{IGBT}$ from previous iteration value and $\boldsymbol{\Delta V}^{IGBT}$.

Algorithm 3 IGBT Kernel

```
procedure PHYSICS BASED IGBT MODULE
    Check PN junction voltage v_eb(t) and v_bc(t) from last iteration value      ▷ Kernel_0
    Check MOSFET junction voltage v_gs(t) from last iteration value             ▷ Kernel_1
Update paramters from IGBT-AE to IGBT-DLE
    Solve p_0 as (3.45) to (3.47)                                               ▷ Kernel_2
    Calculate Q, Q_1 and Q_2 as(3.42) to (3.44)                                 ▷ Kernel_3
    Update intermediate parameter M as (3.53)                                   ▷ Kernel_4
    Calculate C and Q in C_gd, C_gs, C_dsj and C_eb as (3.55) to (3.63)         ▷ Kernel_5
    Calculate C and Q in C_cer and C_mult as (3.65) to (3.66)                   ▷ Kernel_7
    Update I_Q, I_Qeq and G_Q in all capacitors as 3.73 to 3.75                 ▷ Kernel_8
    Calculate r_b as (3.48)                                                     ▷ Kernel_6
    Calculate G_I and I_Ieq for all current sources as (3.39) to (3.72 )        ▷ Kernel_9
    Build matrix equation (3.90) using (3.84) to (3.96) and solve for ΔV        ▷ Kernel_10
    Update V^IGBT(t) for current iteration                                      ▷ Kernel_11
    Check convergence of ΔV^IGBT
    if ΔV^IGBT converges then
        Store V^IGBT to global memory and update t
    else
        Start from checking junction iterative limitation
```

3.3 Newton-Raphson Iteration

3.3.1 Algorithm

Newton-Raphson method is widely used to solve nonlinear equations numerically based on the idea of linear approximation. To find the root of a nonlinear system $\boldsymbol{F}(X)$ consisting k unknown variables, adopting the Jacobian matrix $\boldsymbol{J}_F(X)$ for linear approximation gives following equation,

$$0 = \boldsymbol{F}(\boldsymbol{X}^n) + \boldsymbol{J}_F(\boldsymbol{X}^n)(\boldsymbol{X}^{n+1} - \boldsymbol{X}^n). \tag{3.97}$$

Jacobian matrix $\boldsymbol{J}_F(\boldsymbol{X})$ is a $k \times k$ matrix of all first-order partial derivatives of $\boldsymbol{F}$.

$$\boldsymbol{J}_F = \frac{d\boldsymbol{F}}{d\boldsymbol{X}} = \begin{bmatrix} \frac{\partial F_1}{\partial X_1} & \cdots & \frac{\partial F_1}{\partial X_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_k}{\partial X_1} & \cdots & \frac{\partial F_k}{\partial X_k} \end{bmatrix}. \tag{3.98}$$

Solving for root of $\boldsymbol{F}(X)$ is numerically replaced by solving (3.97) for $\boldsymbol{X}_{n+1} - \boldsymbol{X}_n$ and update $\boldsymbol{X}_{n+1}$ from $\boldsymbol{X}_n$ and the difference until the norm of the difference, $||\boldsymbol{X}_{n+1} - \boldsymbol{X}_n||$, is smaller than a predefined convergence criteria ϵ.

Newton-Raphson method is usually applied in nonlinear power electronic circuits to

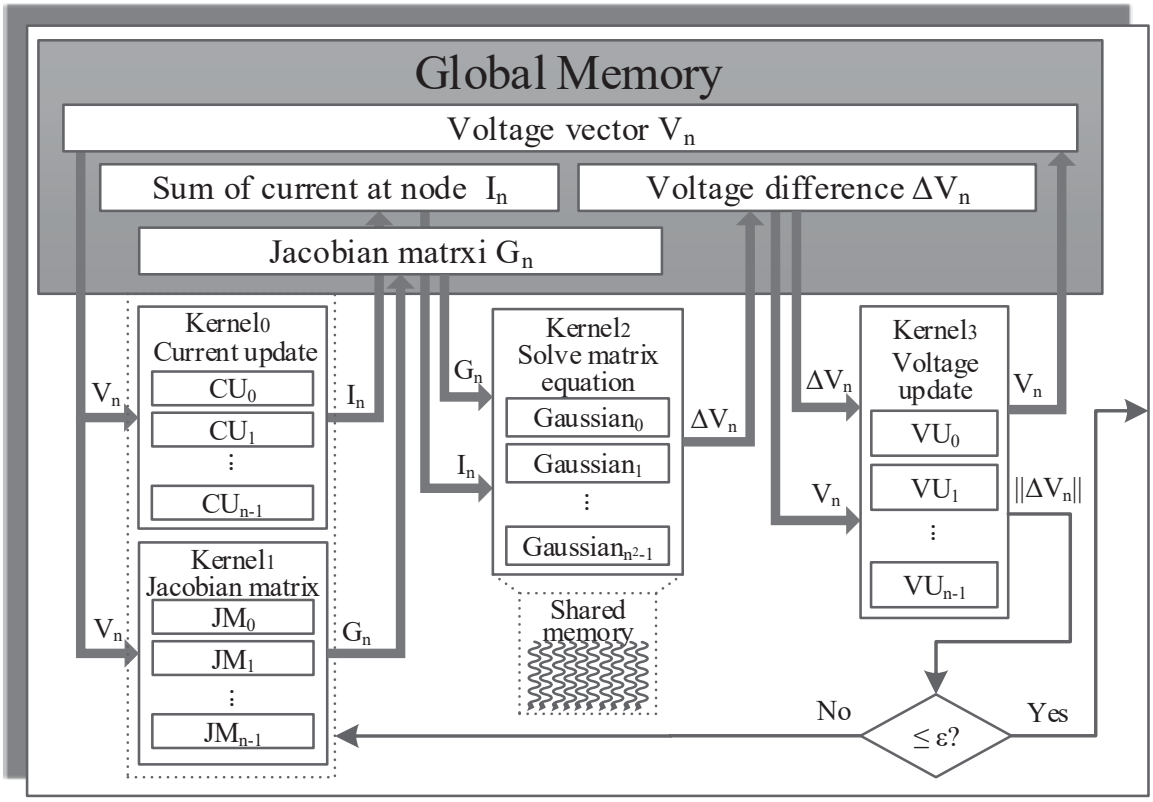


Figure 3.8: Massive thread parallel implementation of Newton-Raphson iteration

solve the transient and stable situations. According to KCL,

$$\Sigma \boldsymbol{I}(\boldsymbol{V}) = 0, \tag{3.99}$$

where $\boldsymbol{I}(\boldsymbol{V})$ refers to the sum of current leaving each node, applying (3.97) gives,

$$\boldsymbol{J}_V{}^n(\boldsymbol{V}^{n+1} - \boldsymbol{V}^n) = -\Sigma \boldsymbol{I}^n, \tag{3.100}$$

Given the physics-based IGBT as an example, $\boldsymbol{J}_V{}^n$ is the $\boldsymbol{G}^{IGBT^{(n)}}$ matrix. Solve the KCL at each node of collector, gate, anode, drain and emitter. In this way, (3.81) gets its Newton-Raphson iterative equation as,

$$\boldsymbol{G}^{IGBT^n}(\boldsymbol{V}^{n+1} - \boldsymbol{V}^n) = -\boldsymbol{I}^n, \tag{3.101}$$

$$-\boldsymbol{I}^n = [i^{c(n)} \quad i^{g(n)} \quad i^{a(n)} \quad i^{d(n)} \quad i^{e(n)}]^T, \tag{3.102}$$

3.3.2 Massive-thread parallel implementation

To develop the massive-thread parallel module for N-R method, 4 kernels are involved as shown in the following algorithm as shown in Fig. 3.8.

Algorithm 4 Newton-Raphson Kernel

```
procedure N-R ITERATION
Iterative loop:
    Calculate F(X^n)                                          ▷ Kernel_0
    Calculate J_F(X^n) from X^n                               ▷ Kernel_1
    Copy J_F(X^n) and F(X^n) into shared memory
    Solve ΔX^n in J_F(X^n)ΔX^n = −F(X^n)                      ▷ Kernel_2
    Update X^{n+1} ← X^n + ΔX^n
    Calculate ||ΔX^n||                                        ▷ Kernel_3
    Store ||ΔX^n|| into global memory
    if ||ΔX^n|| < ε then
        t ← t + Δt
    else
        go to Iterative loop
```

Firstly, Kernel$_0$ is to update $\boldsymbol{F}(\boldsymbol{X}^n)$, in power electronic circuits case, $\boldsymbol{I}^n$ which equals to the sum of currents leaving the nodes. The Jacobian matrix $\boldsymbol{J}_F(\boldsymbol{X}^n)$ which is the admittance matrix in power electronic circuits, is calculated in Kernel$_1$ and copied to shared memory for Kernel$_2$ to solve the equation (3.97). $\boldsymbol{X}^{n+1}$ is updated and verified if the difference is within convergence criteria in Kernel$_3$. The loop will repeat until converged or report convergence error if the number of iterations exceed limitation.

3.4 Gaussian Elimination and LU Decomposition

In the previous Newton-Raphson method, there is a necessity to find an efficient way to solve the linear system in the form of

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}. \tag{3.103}$$

Although there are a lot of ways such as Jocobi method, Gauss-Seidel method, conjugate gradient algorithm to solve the equation. However, these iterative methods are not equipped with the characters to be well paralleled. Jocobi method has relatively low convergence rate depending on the starting value. And in both Gauss-Seidel method and conjugate gradient algorithm, the updating of $\boldsymbol{X}_k$ is dependent on $\boldsymbol{X}_{k-1}$ within an iteration. Take parallelism into consideration, direct methods are preferred for efficiency.

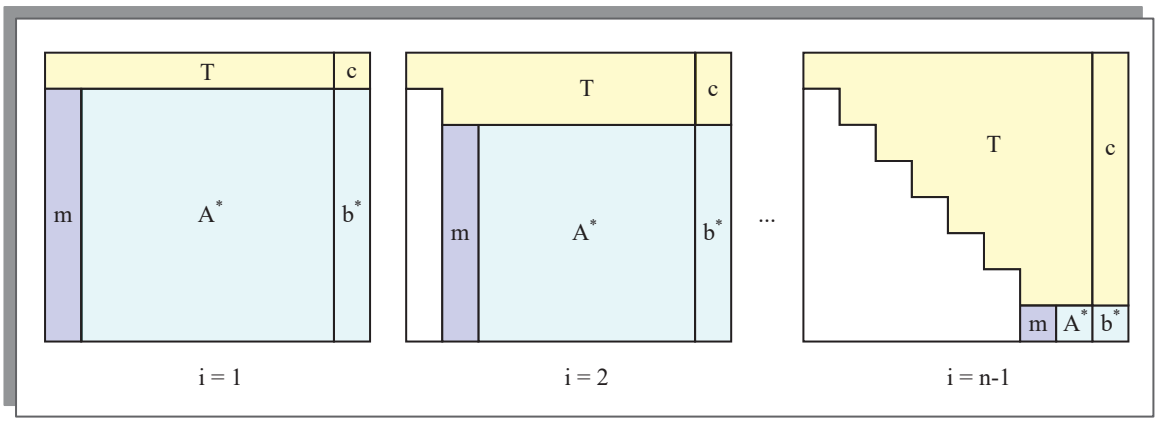


Figure 3.9: Gaussian elimination

3.4.1 Gaussian Elimination Matrix Equation Solution

3.4.1.1 Method formulation

Gaussian elimination matrix equation solution (GEMES) is a direct method as following,

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{3.104}$$

The original Gaussian elimination is to use row reducing method to update the augmented matrix $\boldsymbol{A}^*$ containing $\boldsymbol{A}$ and $\boldsymbol{b}$ to an upper triangular matrix as following,

$$A^*_{i,j} = A^*_{i,j} - A^*_{i,k}/A^*_{k,k} * A^*_{k,j} \qquad (1 \le k \le n-1, k+1 \le i \le n, k+1 \le j \le n+1). \tag{3.105}$$

The parallelism of this method is shown in Fig. 3.9. Each time, column vector $\boldsymbol{m}$ is updated to store the factors as following, since after each step, one column in $\boldsymbol{A}^*$ matrix is calculated to zero, the factor vector $\boldsymbol{m}$ can be updated using zero elements' space in $\boldsymbol{A}^*$ to register.

$$m_i = A^*_{i,k} = A^*_{i,k}/A^*_{k,k} \qquad (1 \le k \le n-1, k+1 \le i \le n). \tag{3.106}$$

The elements in augmented matrix are updated as following,

$$A^*_{i,j} = A^*_{i,j} - A^*_{i,k}A^*_{k,j} \qquad (1 \le k \le n-1, k+1 \le i \le n, k+1 \le j \le n+1). \tag{3.107}$$

(3.106) and (3.107) can be paralleled in GPU, speeding up compared with sequential programming. To ensure convergence and precision, partial pivoting is adopted to find the

maximum elements in column and exchange rows. A permutation matrix $\boldsymbol{P}$ represents the transformation of partial pivoting and (3.103) becomes

$$\boldsymbol{P} \cdot \boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{P} \cdot \boldsymbol{b}. \tag{3.108}$$

After getting the upper triangular augmented matrix, backward substitution is used to compute result vector $\boldsymbol{x}$. Assuming the n×n $\boldsymbol{A}$ matrix together with vector $\boldsymbol{b}$ is converted to upper triangular matrix $\boldsymbol{T}$ and vector $\boldsymbol{c}$, and $\boldsymbol{T}$ and $\boldsymbol{c}$ are stored in the augmented matrix $\boldsymbol{A}^*$ as

$$\boldsymbol{A}^* = [\boldsymbol{T}|\boldsymbol{c}]. \tag{3.109}$$

To solve the equation

$$\boldsymbol{T} \cdot \boldsymbol{x} = \boldsymbol{c}, \tag{3.110}$$

after computing x_k, the products related to x_k are subtracted to update the right side of equation as following,

$$A_{k,n+1} = A_{k,n+1}/A_{k,k} \qquad (n \geq k \geq 1), \tag{3.111}$$

$$A_{i,n+1} = A_{i,n+1} - A_{i,k}A_{k,n+1} \qquad (1 \leq i \leq k-1), \tag{3.112}$$

3.4.1.2 Massive thread parallel implementation

As shown in Fig. 3.10, there are two kernels involved when implementing GEMESM into massive thread parallel module using following algorithm. Kernel_0 is to transfer the augmented matrix $[\boldsymbol{A}|\boldsymbol{b}]$ into upper triangular including partial pivoting. There are 4 steps inside Kernel_0 including finding maximum column element index, row elements swapping, factor column vector updating and minor matrix updating for n×n threads in each block. After coping the augmented matrix $[\boldsymbol{A}|\boldsymbol{b}]$ to shared memory, the row index of maximum elements in target column is found after $\log_2$n times comparison. Then the row elements of the same row index are swapped with the first unfinished row. After row elements swapping, first column elements are updated to store the factors and used to update the rest of the matrix in every thread. This loop will repeat n-1 times and results an upper triangular matrix $[\boldsymbol{T}|\boldsymbol{c}]$ storing in the same position as $[\boldsymbol{A}|\boldsymbol{b}]$. Backward substitution is accomplished in Kernel_1, after n loops to get results from x_n to x_1.

3.4.2 LU Decomposition Matrix Equation Solution

LU Decomposition Matrix Equation Solution (LUDMES) uses LU factorization with partial pivoting decomposes the matrix A into product of two triangular matrix U and L as

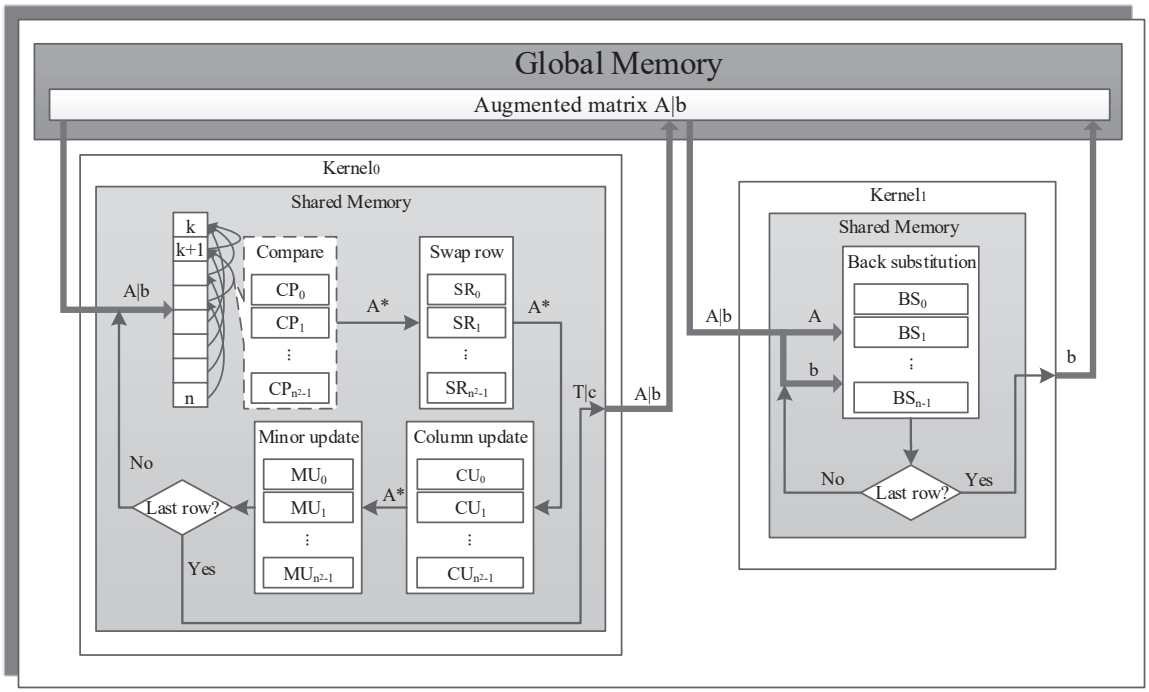


Figure 3.10: Massive thread parallel implementation of GEMESM

Algorithm 5 Gaussian Elimination Kernel

procedure SOLVE MATRIX EQUATION USING GAUSSIAN ELIMINATION

`Upper triangular matrix:`

Find the address of maximum elements in column
Swap row elements with first unfinished row
Column elements update using (3.106)
Update rest of the matrix using(3.107)
if Not last row **then**
  go to `Upper triangular matrix`
else
  Store $[\boldsymbol{T}|\boldsymbol{c}]$ matrix as updated $[\boldsymbol{A}|\boldsymbol{b}]$ into global memory

$\triangleright\ Kernel_0$

`Backward substitution:`

Calculate x_k using (3.111)
Update rest $A_{i,n+1}$ using (3.112)
if Not all x elements calculated **then**
  go to `Backward substitution`
else
  Store final result vector $\boldsymbol{x}$ into global memory as updated $\boldsymbol{b}$

$\triangleright\ Kernel_1$

following,

$$\boldsymbol{P} \cdot \boldsymbol{A} = \boldsymbol{L} \cdot \boldsymbol{U}, \tag{3.113}$$

where $\boldsymbol{P}$ is the row permutations matrix and $\boldsymbol{L}, \boldsymbol{U}$ refer to the lower and upper triangular matrices. If $\boldsymbol{L}$ has all the diagonal elements setting to 1, similar to Gaussian elimination to

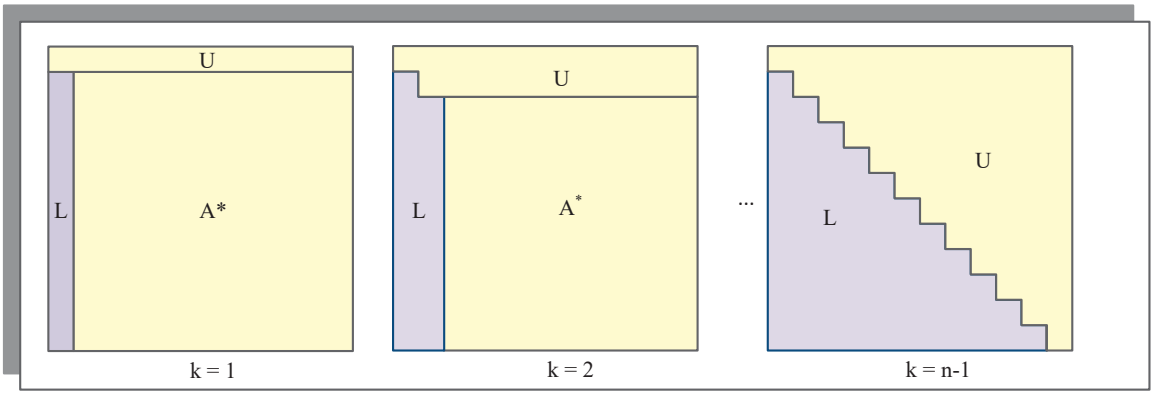


Figure 3.11: LU decomposition

store the factor for current column, elements in each column of $\boldsymbol{L}$ is given as

$$L_{ik} = A_{ik}/A_{kk} \qquad (1 \leq k \leq n-1, k+1 \leq i \leq n). \tag{3.114}$$

The sub-matrix in A is updated to $\boldsymbol{U}$ elements as following,

$$U_{ij} = A_{ij} - L_{ik}A_{kj} \qquad (1 \leq k \leq n-1, k+1 \leq i \leq n, k+1 \leq i \leq n). \tag{3.115}$$

LU decomposition, (3.103) becomes

$$\boldsymbol{L} \cdot \boldsymbol{U} \cdot \boldsymbol{x} = \boldsymbol{P} \cdot \boldsymbol{b}. \tag{3.116}$$

After LU decomposition, if define

$$\boldsymbol{U} \cdot \boldsymbol{x} = \boldsymbol{y}, \tag{3.117}$$

substitution (3.117) into (3.116) yields

$$\boldsymbol{L} \cdot \boldsymbol{y} = \boldsymbol{P} \cdot \boldsymbol{b}. \tag{3.118}$$

To solve vector $\boldsymbol{x}$, there are two step after LU decomposition, forward substitution to solve $\boldsymbol{y}$ in (3.118) and backward substitution to solve $\boldsymbol{x}$ in (3.117). Similar to the backward substitution when use Gaussian elimination to solve matrix equation, forward substitution for $\boldsymbol{y}$ is given as

$$y_k = (\boldsymbol{Pb})_k / L_{k,k} \qquad (1 \leq k \leq n), \tag{3.119}$$

$$y_i = (\boldsymbol{Pb})_i - L_{i,k}y_k \qquad (k+1 \leq i \leq n). \tag{3.120}$$

Backward substitution to solve $\boldsymbol{x}$ is as

$$x_k = y_k / U_{k,k} \qquad (n \geq k \geq 1), \tag{3.121}$$

$$x_i = y_i - U_{i,k}y_k \qquad (1 \leq i \leq k-1). \tag{3.122}$$

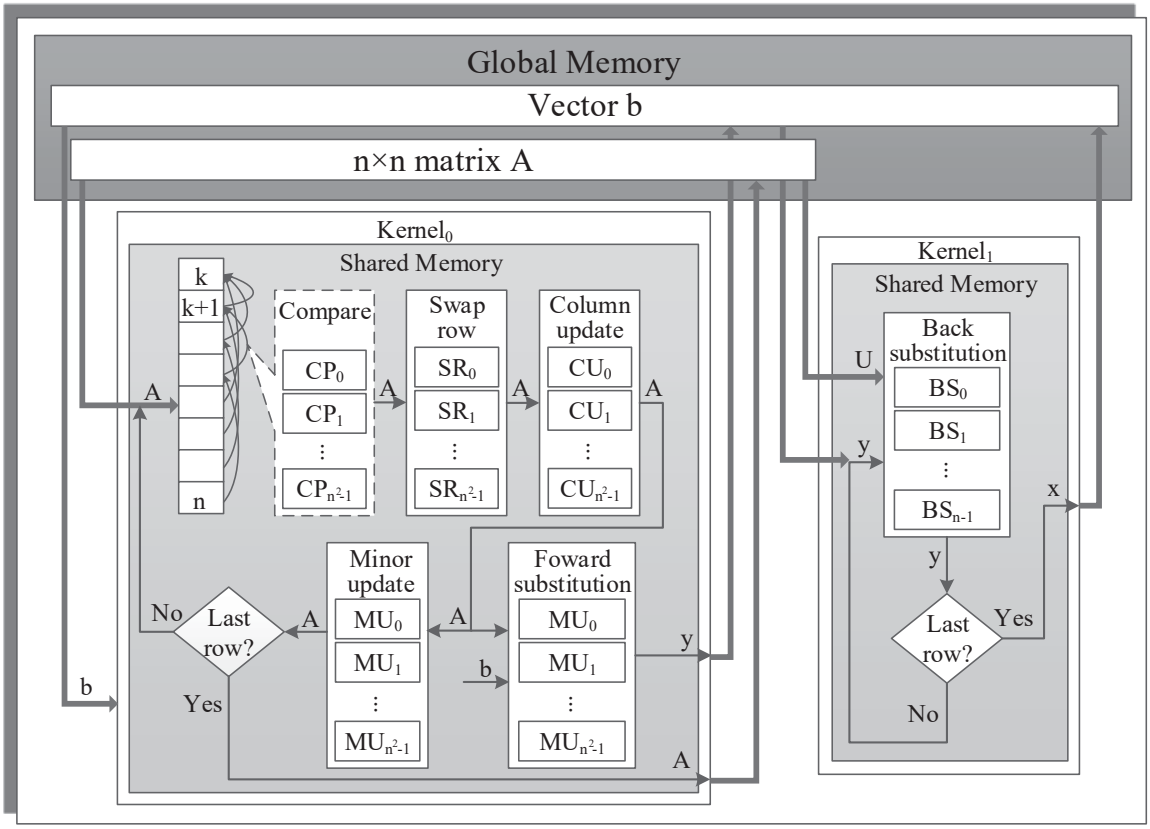


Figure 3.12: Massive thread parallel implementation of LUDMESM

3.4.2.1 Massive Thread Parallel Implementation

The procedure of solving matrix equation using LU decomposition is similar to Gaussian elimination except the following features,

- GEMESM transfers the n×n matrix A into an upper triangular matrix along with the vector b while LUDMESM only deals with the A in to upper and lower triangular matrix L and U first. Therefore, in the cases that Jacobian matrix stays the same during N-R iteration and only vector b changes, LU decomposition will be preferred.
- After transferring A matrix and vector b, there is only backward substitution to do for GEMESM while there is forward substitution followed by backward substitution for LUDMESM.

As shown in Fig. 3.12, the massive thread parallel module for LUDMESM contains two kernels. The LU decomposition and forward substitution are accomplished in $Kernel_0$. And backward substitution is accomplished in $Kernel_1$. The forward substitution in LUDMESM is done simultaneously with the minor of matrix updating. To avoid data transportation

between shared memory and global memory frequently, tasks utilizing the same number of threads are combined to be done in one kernel. Compared with GEMESM, though LUDMESM has one more forward substitution to do, it can be done while doing the triangular decomposition.

Algorithm 6 LU decomposition

```
procedure SOLVE MATRIX EQUATION USING LU DECOMPOSITION
LU decomposition:
    Find the address of maximum elements in column
    Swap row elements with first unfinished row
    Column elements update using (3.106)
    Update rest of the matrix using(3.107)
Forward substitution:
    Calculate y_k using (3.119)                                  ▷ Kernel_0
    Update y_i using (3.120)
    if Not last row then
        go to LU decomposition
    else
        Store L and U matrix and vector y into global memory
Backward substitution:
    Calculate x_k using (3.121)
    Update rest x_i using (3.122)
    if Not all x elements calculated then                        ▷ Kernel_1
        go to Backward substitution
    else
        Store final results into global memory
```

3.5 Block Jacobian Matrix Computation for Modular Multi-level Converter

In Modular Multi-level Converter (MMC) circuit, a serial of IGBTs and power diodes are connected to convert between DC and AC in HVDC system, which means there are more than one nonlinear elements in one node. One IGBT and one diode consist an unit and two units build a sub module (SM). Fig. 3.13 shows the connection inside an SM and the node order. Using the physics-based IGBT and power diode module mentioned above, IGBT has 5 nodes and power diode has one more inside, resulting the Jacobian matrix, which is also the G matrix for the SM has the shape of

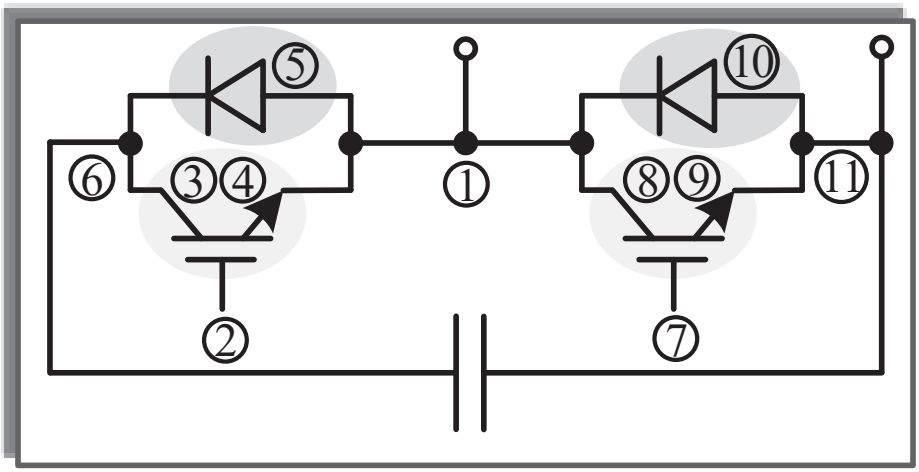


Figure 3.13: Node order of an SM in MMC

$$
\mathrm{G}_{SM}^{original} =
$$

$$
\begin{bmatrix}
G_{1,1} & G_{1,2} & G_{1,3} & G_{1,4} & G_{1,5} & G_{1,6} & G_{1,7} & G_{1,8} & G_{1,9} & G_{1,10} & G_{1,11} \\
G_{2,1} & G_{2,2} & G_{2,3} & G_{2,4} & G_{2,5} & G_{2,6} & & & & & \\
G_{3,1} & G_{3,2} & G_{3,3} & G_{3,4} & G_{3,5} & G_{3,6} & & & & & \\
G_{4,1} & G_{4,2} & G_{4,3} & G_{4,4} & G_{4,5} & G_{4,6} & & & & & \\
G_{5,1} & G_{5,2} & G_{5,3} & G_{5,4} & G_{5,5} & G_{5,6} & & & & & \\
G_{6,1} & G_{6,2} & G_{6,3} & G_{6,4} & G_{6,5} & G_{6,6} & & & & & G_{6,11} \\
G_{7,1} & & & & & & G_{7,7} & G_{7,8} & G_{7,9} & G_{7,10} & G_{7,11} \\
G_{8,1} & & & & & & G_{8,7} & G_{8,8} & G_{8,9} & G_{8,10} & G_{8,11} \\
G_{9,1} & & & & & & G_{9,7} & G_{9,8} & G_{9,9} & G_{9,10} & G_{9,11} \\
G_{10,1} & & & & & & G_{10,7} & G_{10,8} & G_{10,9} & G_{10,10} & G_{10,11} \\
G_{11,1} & & & & & G_{11,6} & G_{11,7} & G_{11,8} & G_{11,9} & G_{11,10} & G_{11,11}
\end{bmatrix}
\tag{3.123}
$$

3.5.1 Matrix Updating Using a Relaxation Algorithm

Matrix $\mathrm{G}_{SM}^{original}$ has many zero elements, and to make it better organized, the two admittance, $\mathrm{G}_{6,11}$ and $\mathrm{G}_{11,6}$, brought by capacitance are to be transformed using a relaxation algorithm. The Newtone-Raphson equation for a SM is given as

$$
G_{SM}^{original^n} \cdot \Delta V^n = -I^n, \tag{3.124}
$$

which can be transformed as following,

$$
G_{SM}^n \cdot \Delta V^n = -I^n + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -G_{6,11}\Delta v_{11}^n & 0 & 0 & 0 & 0 & -G_{11,6}\Delta v_6^n \end{bmatrix}^T, \tag{3.125}
$$

Where $G_{SM} =$

$$\begin{bmatrix} G_{1,1} & G_{1,2} & G_{1,3} & G_{1,4} & G_{1,5} & G_{1,6} & G_{1,7} & G_{1,8} & G_{1,9} & G_{1,10} & G_{1,11} \\ G_{2,1} & G_{2,2} & G_{2,3} & G_{2,4} & G_{2,5} & G_{2,6} & & & & & \\ G_{3,1} & G_{3,2} & G_{3,3} & G_{3,4} & G_{3,5} & G_{3,6} & & & & & \\ G_{4,1} & G_{4,2} & G_{4,3} & G_{4,4} & G_{4,5} & G_{4,6} & & & & & \\ G_{5,1} & G_{5,2} & G_{5,3} & G_{5,4} & G_{5,5} & G_{5,6} & & & & & \\ G_{6,1} & G_{6,2} & G_{6,3} & G_{6,4} & G_{6,5} & G_{6,6} & & & & & \\ G_{7,1} & & & & & & G_{7,7} & G_{7,8} & G_{7,9} & G_{7,10} & G_{7,11} \\ G_{8,1} & & & & & & G_{8,7} & G_{8,8} & G_{8,9} & G_{8,10} & G_{8,11} \\ G_{9,1} & & & & & & G_{9,7} & G_{9,8} & G_{9,9} & G_{9,10} & G_{9,11} \\ G_{10,1} & & & & & & G_{10,7} & G_{10,8} & G_{10,9} & G_{10,10} & G_{10,11} \\ G_{11,1} & & & & & & G_{11,7} & G_{11,8} & G_{11,9} & G_{11,10} & G_{11,11} \end{bmatrix} \tag{3.126}$$

Since capacitances in the circuit are big and there is an outer Newton-Raphson loop, relaxation algorithm can be adopted to use previous value Δv_6^{n-1} and Δv_{11}^{n-1} instead of current ones. In this way, updating the right side of equation (3.125) using known values results a better block organized $\boldsymbol{G}_{SM}$.

3.5.2 Partial LU Decomposition for Block Jacobian Matrix

3.5.2.1 Partial LU Decomposition for a Single Submodule

To solve the equation (3.125), a method utilizing G_{SM}'s block character called partial LU decomposition is adopted to solve the specific case. Given an example of situation containing only one SM, the Jacobian matrix is exactly G_{SM}. If denote each block of G_{SM} as shown in Fig. 3.14, where A_1 and A_2 are 5×5 matrices, c_1 and c_2 are 1×5 row vectors, d_1 and d_2 are 5×1 column vectors and e is only one element.

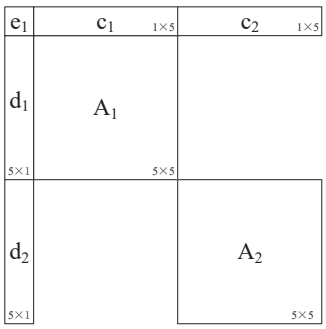


Figure 3.14: Jacobian matrix of a SM

Applying LU decomposition to A_1 and A_2 yields the following equations,

$$L_1 \cdot U_1 = A_1, \tag{3.127}$$

$$L_2 \cdot U_2 = A_2. \tag{3.128}$$

Define that partial LU decomposition of G_{SM} shown in Fig. 3.15,

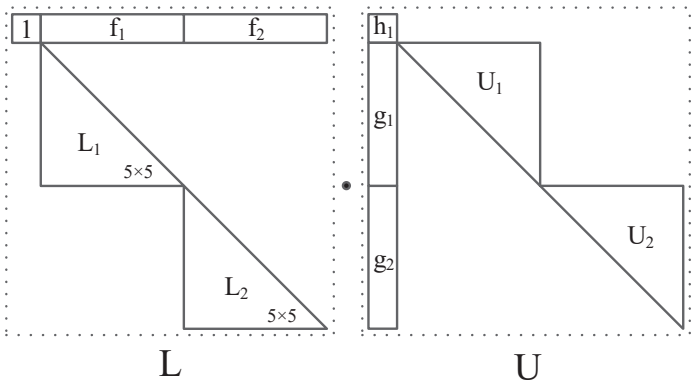


Figure 3.15: Partial LU decomposition of a SM Jacobian matrix

where f_1,f_2,g_3,g_4,h_1 satisfy the following relationship,

$$f_1 \cdot U_1 = c_1 \tag{3.129}$$

$$f_2 \cdot U_2 = c_2 \tag{3.130}$$

$$L_1 \cdot g_1 = d_1 \tag{3.131}$$

$$L_2 \cdot g_2 = d_2 \tag{3.132}$$

$$h_1 + f_1 \cdot g_1 + f_2 \cdot g_2 = c_1. \tag{3.133}$$

Once L_1,U_1, L_2 and U_2 are computed using LU decomposition as (3.127) to (3.128), f_1,f_2,g_3,g_4 can be calculated using backward and forward substitution in (3.129) to (3.132), followed by getting h_1 according to (3.133). This partial LU decomposition computes the semi-upper-triangular matrix L and semi-lower-triangular matrix U and the decomposition procedure is able to be parallelized.

After obtaining the L and U matrices, it is similar to forward and backward substitution in solving matrix equation using original LU decomposition while nearly twice faster. To solve the equation

$$L \cdot U \cdot x = b, \tag{3.134}$$

define

$$U \cdot x = y, \tag{3.135}$$

Vector y is divided into 3 parts denoted as y_1, y_2 and y_3, which has the size length of 1, 5 and 5. Similar method is applied to vector x and vector b into 3 parts. And y_2 and y_3 are to be computed in parallel using forward substitution, since

$$L_1 \cdot y_2 = b_2 \tag{3.136}$$

$$L_2 \cdot y_3 = b_3, \tag{3.137}$$

then

$$y_1 = b_1 - f_1 \cdot y_2 - f_2 \cdot y_3. \tag{3.138}$$

Similarly, to solve

$$U \cdot x = y, \tag{3.139}$$

x_1 can be solved directly as

$$x_1 = y_1/b_1, \tag{3.140}$$

y_2 and y_3 are updated to subtract the x_1 part as following,

$$y_2' = y_2 - x_1 g_1 \tag{3.141}$$

$$y_3' = y_3 - x_1 g_2. \tag{3.142}$$

Using backward substitution can solve x_2 and x_3 individually according to

$$U_1 \cdot x_2 = y_2 \tag{3.143}$$

$$U_2 \cdot x_3 = y_3. \tag{3.144}$$

3.5.2.2 Partial LU Decomposition for MMC

When there is only one SM inside a circuit, if one IGBT and one diode are grouped as a pair of nonlinear elements, only two pairs of nonlinear elements are connected in one node. For a complete MMC circuit, there are nodes connecting three pairs of nonlinear elements. The complete Jacobian matrix G_{MMC} has the structure shown in Fig. 3.16.

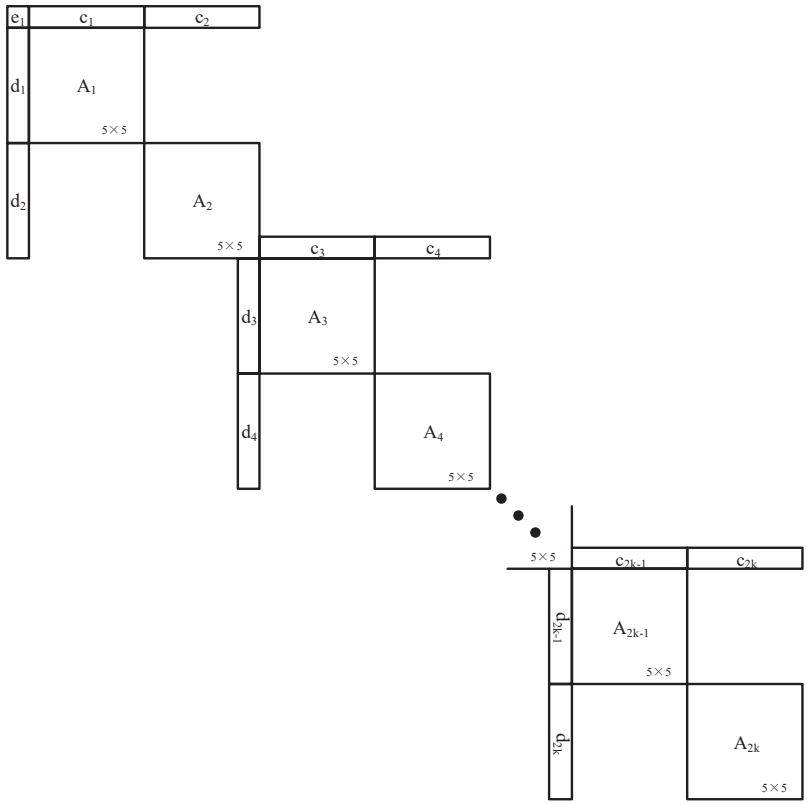


Figure 3.16: Jacobian matrix structure of MMC circuit

For convenience to latter decomposition, the connection between node 1 and node 11 are to be simplified using the method as dealing with the capacitance inside a SM using the relaxation algorithm. Similar to the node 11 and 21, ... To solve the equation

$$G^{n}_{MMC} \cdot \Delta V^{n} = -I^{n} \tag{3.145}$$

is equal to solve the following equation

$$G^{n*}_{MMC} \cdot \Delta V^{n} = (-I)^{n*}, \tag{3.146}$$

where G^{n*}_{MMC} is updated as shown in Fig. 3.17.

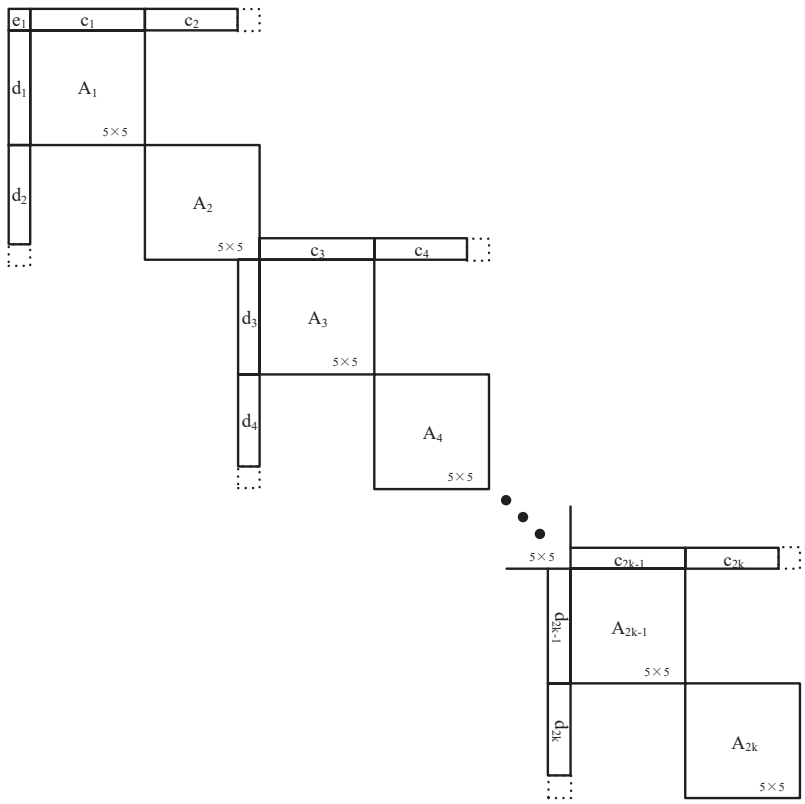


Figure 3.17: MMC updated Jacobian Matrix G^{n*}_{MMC} using the relaxation algorithm

$$\begin{aligned}(-I)^{n*} =&(-I)^{n} + (-G_{MMC1,11}\Delta V_{11}^{n-1} - G_{MMC11,1}\Delta V_{1}^{n-1} - G_{MMC11,21}\Delta V_{21}^{n-1} \\ &- G_{MMC21,11}\Delta V_{11}^{n-1} - ... - G_{MMC10k-9,10k+1}\Delta V_{10k+1}^{n-1} \\ &- G_{MMC10k+1,10k-9}\Delta V_{10k-9}^{n-1})\end{aligned} \tag{3.147}$$

Similar to the method for a single SM, partial LU decomposition can still be applied with some improvement as shown in Fig. 3.18. Complete LU decomposition to A_1, A_2...A_{2k-1} individually in a MMC circuit containing k SMs can be fully parallelized. And f_1, f_2...f_{2k-1} and g_1, g_2...g_{2k-1} is calculated using backward and forward substitution. The product of semi-upper-triangular matrix L and semi-lower-triangular matrix U as following is almost G^{*}_{MMC}.

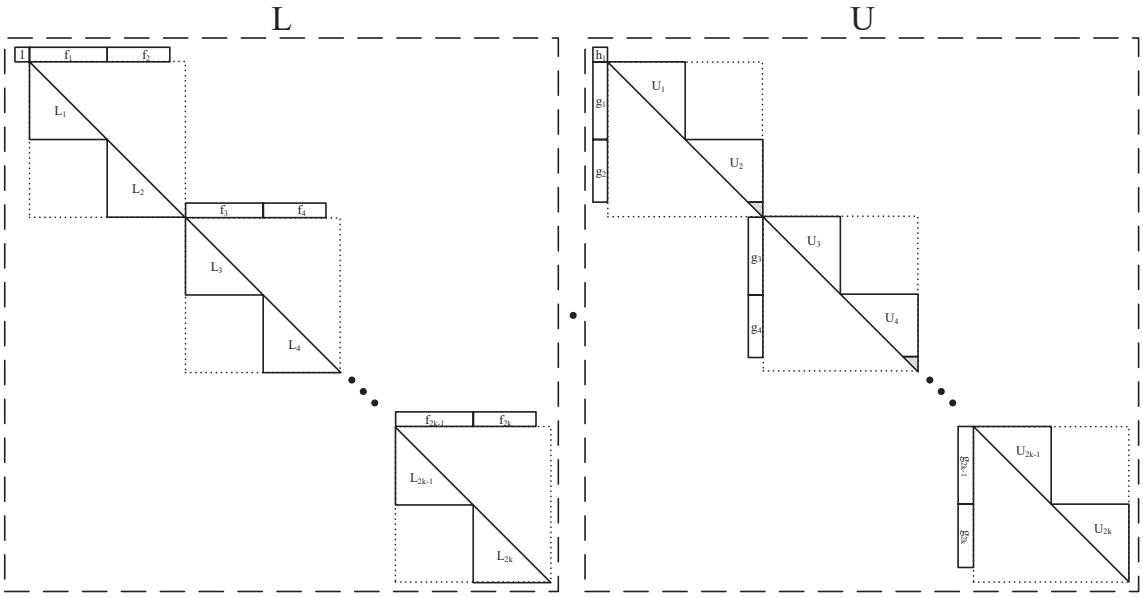


Figure 3.18: Partial LU decomposition for G^*_{MMC}

All the elements in the product, named as P, are equal to the ones in G^*_{MMC} except the ones in connection with two SMs. Given $P_{11,11}$ as an example, is equal to the product of last row of L_1 and last column of U_2 in addition of $f_3 \cdot g_3 + f_4 \cdot g_4$ however $G^*_{MMC_{11,11}}$ is only the product of last row of L_1 and last column of U_2. This can be accomplished by modify $U_{m_{5,5}}$ to $U^*_{m+1_{5,5}}$ as following,

$$U^*_{m+1_{5,5}} = U_{m+1_{5,5}} - f_{m+2} \cdot g_{m+2} - f_{m+3} \cdot g_{m+3} \qquad (m = 1, 3, ..., 2k-1), \quad (3.148)$$

After getting the L and U matrix utilizing partial LU decomposition, similar to previous method, forward and backward substitution are explained as following. If define

$$U \cdot x = y, \quad (3.149)$$

to solve

$$L \cdot y = b, \quad (3.150)$$

using blocked forward substitution as in Fig. 3.19, vector y is divided into small vectors, numbering each size 5 vector from y_1 to y_{2k}.

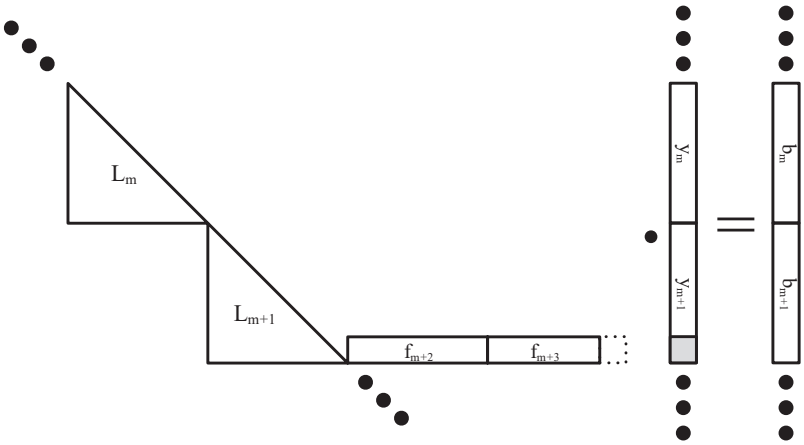


Figure 3.19: Blocked forward substitution

Since

$$L_m \cdot y_m = b_m \qquad (m = 1, 3, ..., 2k-1), \tag{3.151}$$

$$L_{m+1} \cdot y_{m+1} + f_{m+2} \cdot y_{m+2} + f_{m+3} \cdot y_{m+3} = b_{m+1} \qquad (m = 1, 3, ..., 2k-1), \tag{3.152}$$

forward substitution for each block as $L_m \cdot y_m = b_m$ gives y_m individually. After that, y_m and first 4 elements in y_{m+1} are already calculated to satisfy (3.19) and there is one more step to update one element y_{m+1_5} as following,

$$y_{m+1_5} = y_{m+1_5} - f_{m+2} \cdot y_{m+2} - f_{m+3} \cdot y_{m+3}. \tag{3.153}$$

The equation

$$U \cdot x = y \tag{3.154}$$

has the following structure in Fig .3.20.

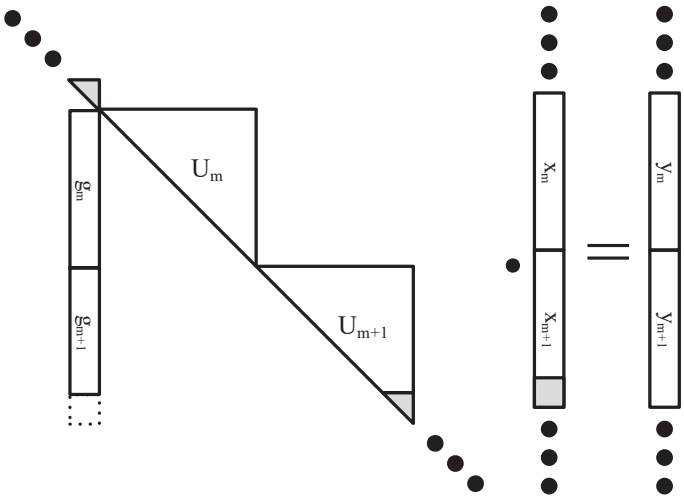


Figure 3.20: Blocked backward substitution

Since

$$U_m \cdot x_m + g_m \cdot x_{m-1_5} = y_m \qquad (m = 1, 3, ..., 2k-1), \tag{3.155}$$

$$U_{m+1} \cdot x_{m+1} + \begin{bmatrix} g_{m+1} \\ 0 \end{bmatrix} \cdot x_{m-1_5} = y_{m+1} \qquad (m = 1, 3, ..., 2k-1), \tag{3.156}$$

x_{m+1_5} can be calculated individually for each m using

$$x_{m+1_5} = y_{m+1_5} / U_{m+1_5}. \tag{3.157}$$

x_{m-1_5}'s effect in y_m and y_{m+1} is subtracted as following

$$y_m = y_m - g_m \cdot x_{m-1_5} \qquad (m = 1, 3, ..., 2k-1), \tag{3.158}$$

$$y_{m+1} = y_{m+1} - \begin{bmatrix} g_{m+1} \\ 0 \end{bmatrix} \cdot x_{m-1_5} \qquad (m = 1, 3, ..., 2k-1). \tag{3.159}$$

Then x_m and x_{m+1} can be computed using normal backward substitution

$$U_m \cdot x_m = y_m \qquad (m = 1, 3, ..., 2k-1), \tag{3.160}$$

$$U_{m+1} \cdot x_{m+1} = y_{m+1} \qquad (m = 1, 3, ..., 2k-1). \tag{3.161}$$

This partial LU decomposition is suitable for parallel simulation, it utilizes character of boarded block diagonal matrix. In MMC circuit, the size of Jacobian matrix grows with the number of output voltage level. In stead of solving an equation containing a (11k-1)×(11k-1) Jacobian matrix, it solves the 5×5 block as a computing unit, in addition, this method utilizes limited share memory in GPU programming.

3.5.3 Parallel Massive-thread Mapping

As shown in Fig. 3.21, there are 4 kernels involved in the parallel module for the partial LU decomposition method in MMC. Kernel_0 is to update the matrix equation using relaxation algorithm. The original G_{MMC} matrix will have the shape of G^*_{MMC} after completing Kernel_0. To calculate the semi-upper triangular matrix L in a MMC circuit containing k SMs, meaning 2k blocks, there are 3 steps inside Kernel_1 listed as following,

- Normal LU decomposition in each block to get L_j and U_j, where j is from 1 to 2k.
- Backward substitution can get f_j from U_j and c_j, forward substitution can get g_j from L_j and d_j in each block
- An element modification in $U_{m_{5,5}}$, where m=1,3,..., 2k-1.

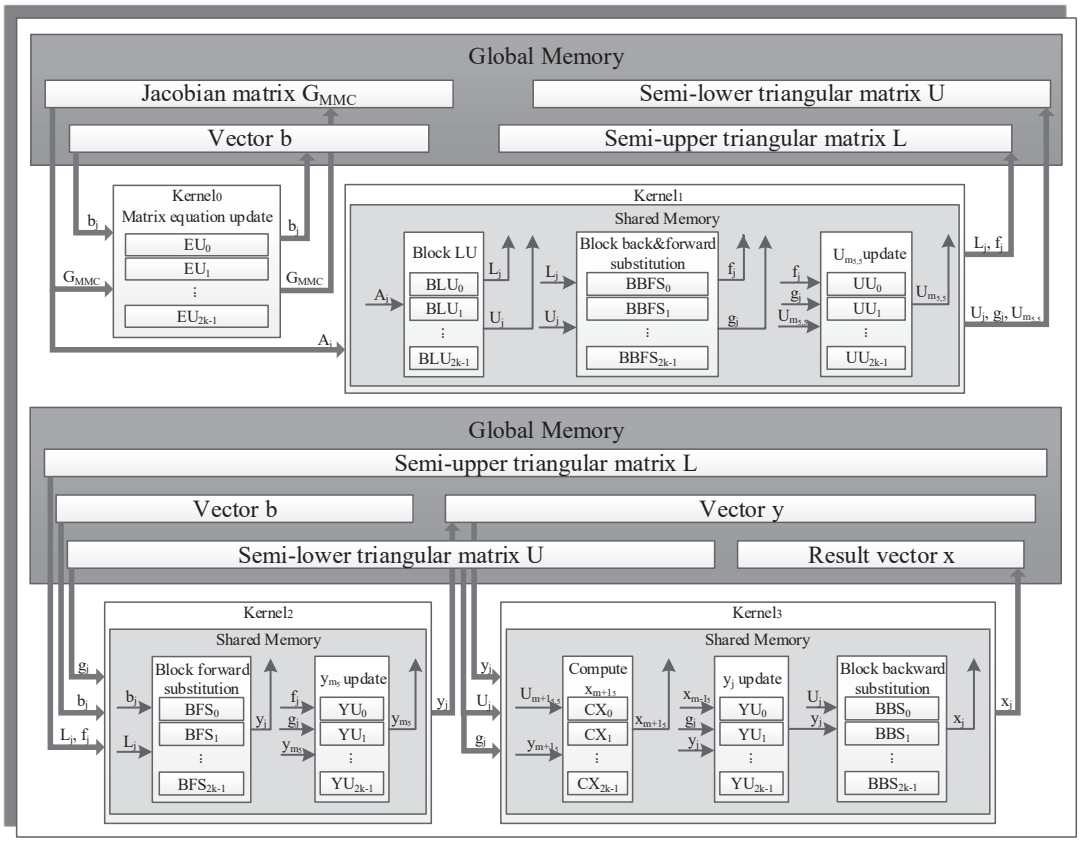


Figure 3.21: Massive thread parallel implementation of partial LU decomposition

Kernel_2 is responsible to calculate vector y similar to forward substitution with additional elements modification. In Kernel_3, the x_{m+1_5} is calculated first and its products are subtracted, followed by backward substitution in each block to get result x_j.

Algorithm 7 Partial LU decomposition for MMC

procedure SOLVE MATRIX EQUATION FOR MMC CIRCUIT USING PARTIAL LU DECOMPOSITION

`Produce semi-upper and semi-lower triangular matrix:`

Update matrix equation using relaxation algorithm (3.125), (3.147) ▷ $Kernel_0$

Block LU decomposition to get L_j, U_j as (3.127), (3.128)
Back and forward substitution for f_j, g_j as (3.129) to (3.132)
$U_{m_{5,5}}$ modification using (3.148) } ▷ $Kernel_1$

`Modified forward substitution:`

Forward substitution in block as (3.151)
y_{m+1_5} update according to (3.153) } ▷ $Kernel_2$

`Modified backward substitution:`

Compute x_{m+1_5} directly as (3.157)
Subtract x_{m-1_5}'s effect to update y_j (3.158), (3.159)
Backward substitution in block as (3.160), (3.161) } ▷ $Kernel_3$

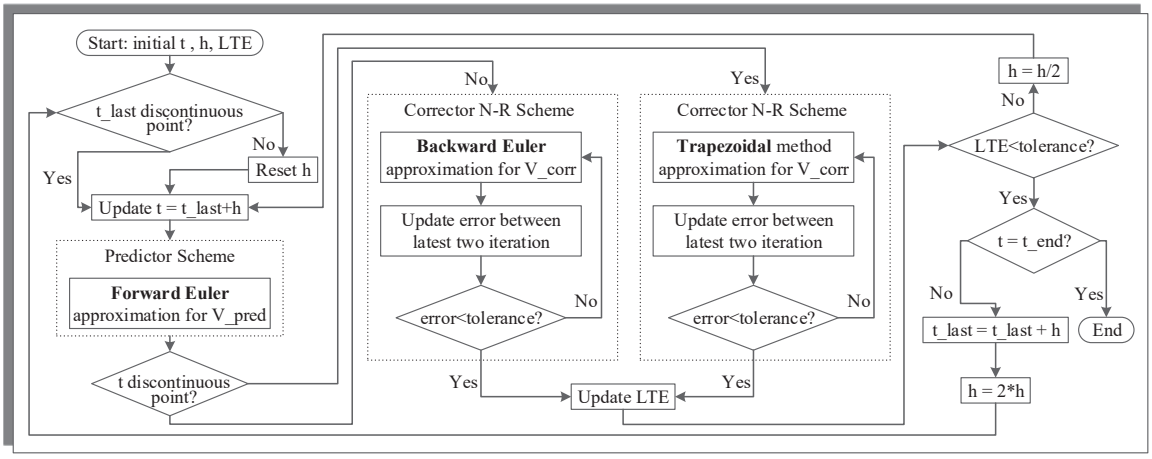


Figure 3.22: Flow chart of Variable Time Step Scheme

3.6 Variable Time-Step Scheme Using Predictor-Corrector Method

A variable time step scheme is used for numerical solution of the transient nonlinear system response to ensure both efficiency and accuracy. In most power electronic systems, transient portion is smaller than steady-state part, dynamically adaptive time step reduces the calculation during steady state while maintains accuracy on transient state.

To find the numerical solution of ordinary differential equations (ODEs), predictor-corrector method offers a way to use a polynomial prediction from the derivative from the previous point for the current point, followed by refining the predicted value. The predictor step usually adopts an explicit method such as Forward Euler method, while the corrector step uses implicit method. If the local truncation error (LTE) is out of tolerance, which means the current time-step size is too large to obtain accurate result, a refined time-step size is adopted to recalculate current time point predictor and corrector.

To solve the system algebraic equation using iterative method, predictor-corrector variable time step method (PCVTSM) is adopted as shown in Fig. 3.22. Based on Gear's method [31], to compute the numerical solution of V(t), forward Euler in this case is used to calculate the approximation from values of t-Δt as a predictor V_{corr}(t). And an implicit method backward Euler or Trapezoidal method depending on if current calculation point is discontinuous to compute a corrector.

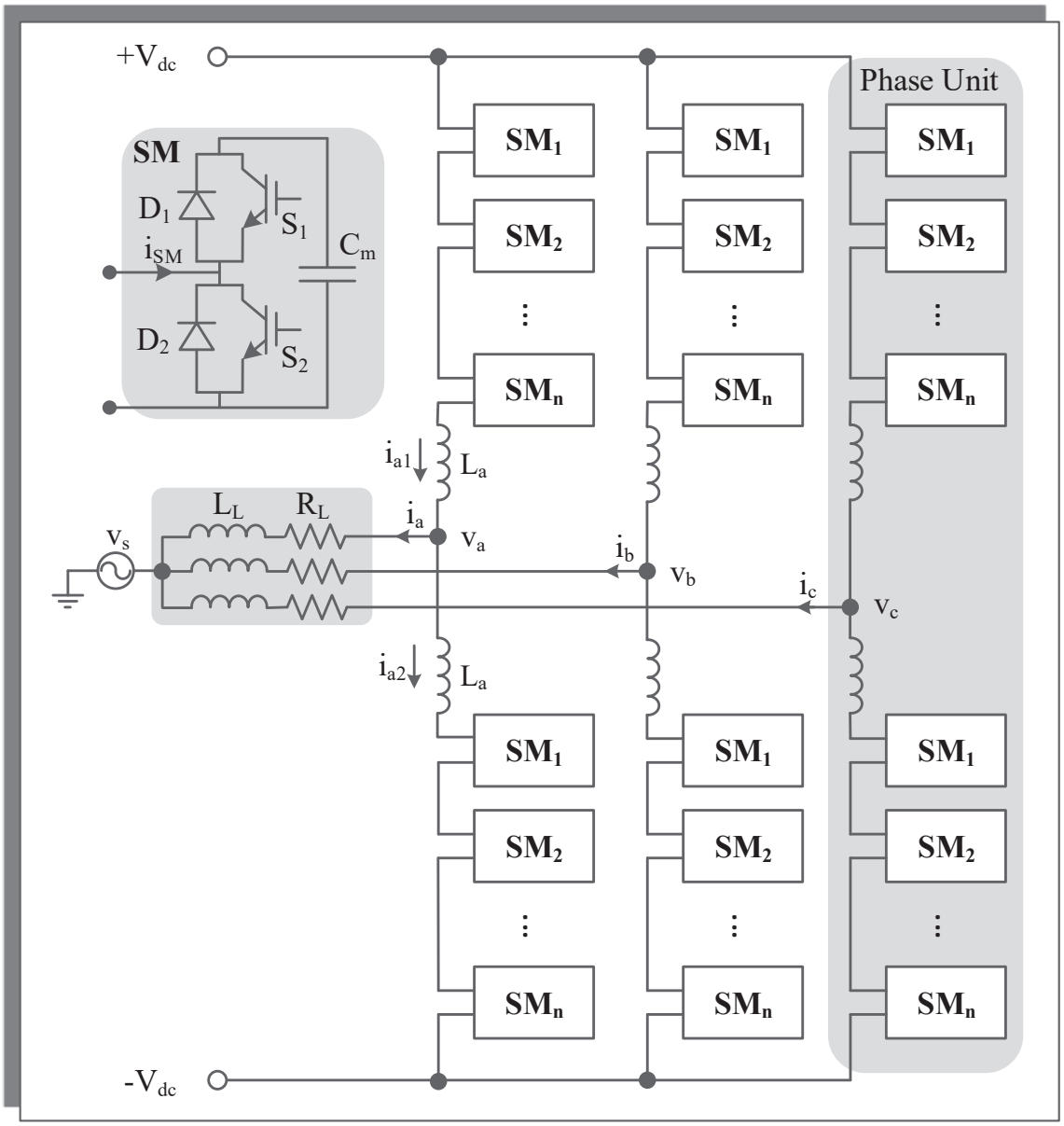


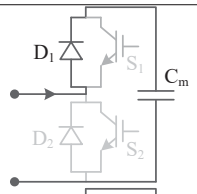
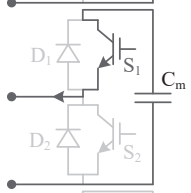
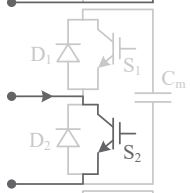
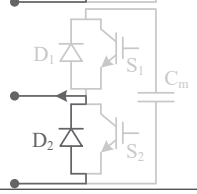
Figure 3.23: MMC circuit structure

3.7 Modular Multi-Level Converter (MMC)

3.7.1 Circuit Structure

MMC circuit has popular application on HVDC circuit, there are different types of simulation models including physics-based models, equivalent circuit base model, arm switching function and average value model [32]- [36]. Even physics-based model provides most detailed information inside every device, to achieve this, time-step size should be in nanosecond, therefore, the computational requirement makes it uncommonly used in

Table 3.1: Capacitor charging and switching state of an SM

Gate Combination	i_{SM}	Capacitor State	SM Operating Figure
10	>0	Charging	
10	<0	Discharging	
01	>0	Unchanged	
01	<0	Unchanged	

simulation [37].

Fig. 3.23 shows a 3-phase cascade MMC circuit structure consisting of n half bridge submodules (SM) in each arm. Each half bridge SM is consisting of two of IGBTs, two inverse-parallel diodes and an energy storage capacitor. Based on the gate signal combination of each IGBT inside SM and current direction, SM has different operating states. In Tab. 3.1, once S_1 gate has on signal and S_2 gate has off signal, C_m will charge and discharge according to i_SM direction. In addition, gate signal combination 00, which means SM is blocked, is not used in normal operation. And gate signal combination 11 will cause the short circuit of the capacitor.

3.7.2 Phase-Shifted Carrier Modulation Strategy

The control strategies of MMC circuit adopted in this thesis include the active and reactive power control, capacitor voltage averaging and balancing control [38]- [40]. In Fig. 3.24, the control loop for power in [41] utilizes the abc/dq0 frame transformation for output ac power calculation.

$$P = v_d i_d + v_q i_q, \tag{3.162}$$

$$Q = v_q i_d - v_d i_q, \tag{3.163}$$

When the voltage vector aligned with the d axis, which means v_q =0, gives the following equation,

$$P = v_d i_d \tag{3.164}$$

$$Q = v_d i_q. \tag{3.165}$$

In Fig. 3.23, the sum of n SMs voltage in upper arm and lower arm are v_{a1} and v_{a2} in phase a to illustrate control strategy, the following equations are obtained,

$$(v_{ca} - v_{sa}) - (\frac{L_a}{2} + L_l)\frac{di_a}{dt} - R_l i_a = 0, \tag{3.166}$$

where v_{ca} and v_{sa} is the reference converter output voltage and point of coupling (PCC) voltages in phase-a. Applying abc/dq frame transformation yields

$$(v_{cd} - v_{sd}) = (\frac{L_a}{2} + L_l)\frac{di_d}{dt} - \omega(\frac{L_a}{2} + L_l)i_q + R_l i_d \tag{3.167}$$

$$(v_{cq} - v_{sq}) = (\frac{L_a}{2} + L_l)\frac{di_q}{dt} + \omega(\frac{L_a}{2} + L_l)i_d + R_l i_q. \tag{3.168}$$

For current control, the following equations are obtained from (3.167) and (3.168),

$$\frac{di_d}{dt} = \frac{v_{cd} - v_{sd} + \omega L_c i_q}{L_c} - \frac{R_l i_d}{L_c}, \tag{3.169}$$

$$\frac{di_q}{dt} = \frac{v_{cq} - v_{sq} - \omega L_c i_d}{L_c} - \frac{R_l i_q}{L_c}, \tag{3.170}$$

where

$$L_c = \frac{L_a}{2} + L_l. \tag{3.171}$$

Based on [42], the outer-loop in Fig. 3.24 is the active power controller while the inner-loop is the current controller. Given the fixed power reference, the difference of active and reactive power is given to PI controllers to produce reference dq current. After producing the dq voltage, the abc phase reference voltage is calculated for modulation.

The capacitor voltage averaging and balanced control are shown in Fig. 3.25, the reference capacitor voltage v_c^{ref} is to be compared with capacitors' average voltage followed by a PI controller to produce a DC loop current command i_Z. The difference between arm current average value i_a and i_Z is to produce the average control voltage v_A through another PI control. For balance control voltage v_B, the capacitor voltage in every SM is to be compared with v_c followed by the gain of K_5. And whether the SM is in the upper

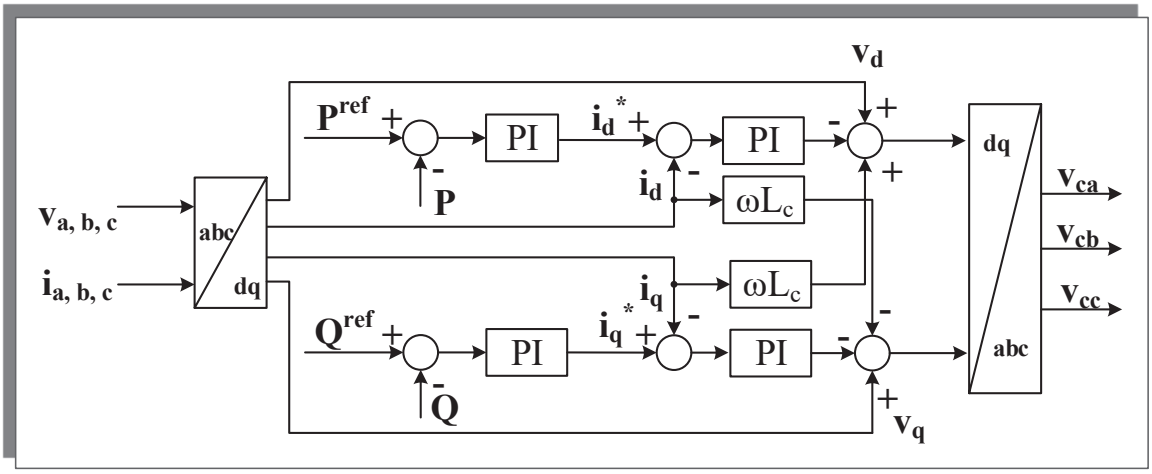


Figure 3.24: Active and reactive power control of the MMC

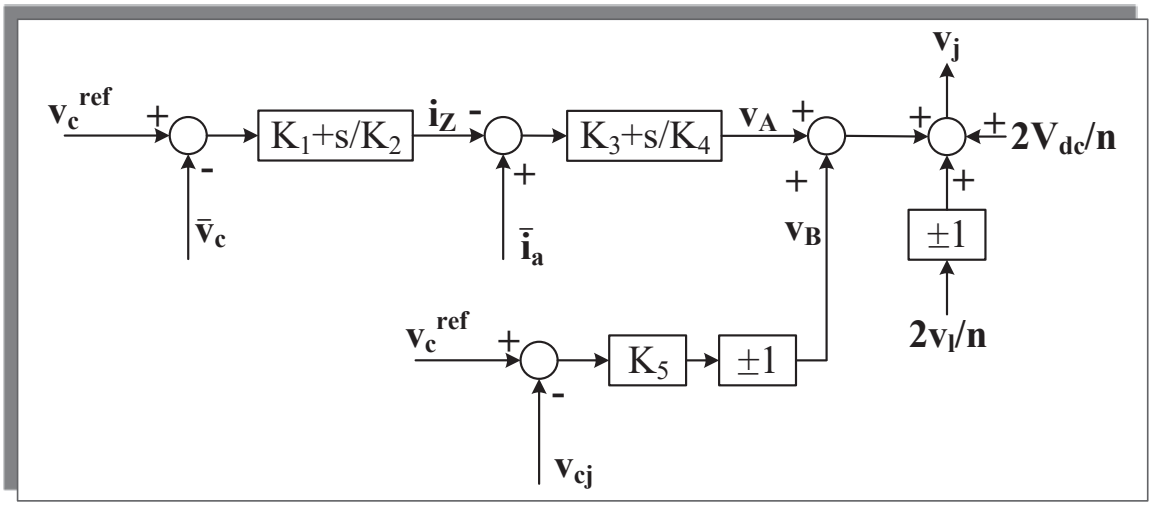


Figure 3.25: Averaging and balancing control of the MMC

or lower arm decide the direction of v_B. After that, the total control signal should take account of load voltage reference v_l and feed forward voltage of V_{dc}. Before comparing with carrier waves, v_j need to be normalized using v_c^{ref}. Averaging control is to make sure average voltage value of all the capacitors in every phase to follow the command value, while the balancing control adopts phase-shifted carrier signals to force the dc voltage of each capacitors to follow the reference value. If there are n SMs in one phase, the normalized modulation signal in Fig .3.26 is compared with n the carrier signals to produce gate signal for each SM. Each carrier signal has a phase shift of 2π /n with each other.

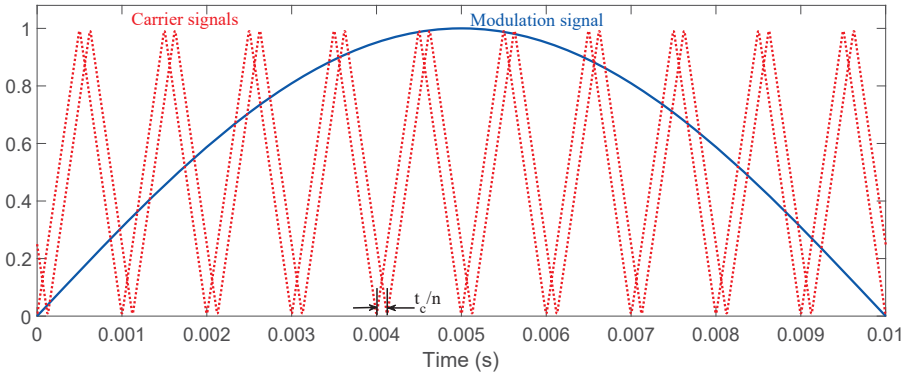


Figure 3.26: Modulation signal and phase shift carrier signals

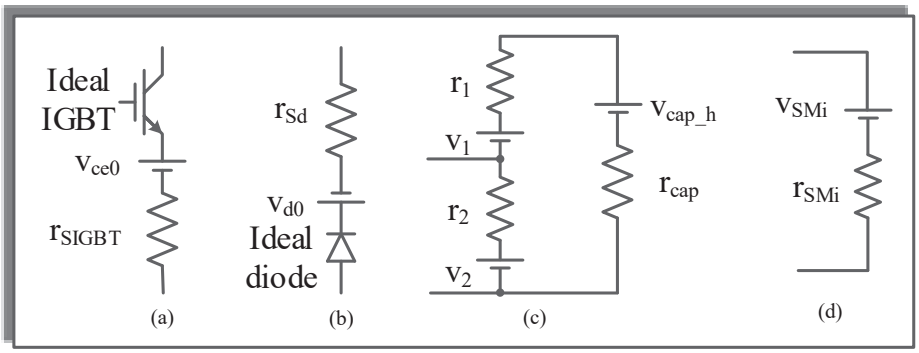


Figure 3.27: (a) Behavior-based IGBT model, (b) Behavior-based power diode model, (c) Discretized SM circuit, (d) Equivalent Thevenin circuit of SM

3.8 Behavior-based MMC Solution

3.8.1 Equivalent Model for SM

In system-level MMC circuit simulation, the behavior-based IGBT and diode models in [43] are commonly adopted. IGBT and diode are modeled as ideal switch models in series with voltage source and resistors, which is shown in Fig. 3.27(a), (b). Based on the fact that the relationship between output voltage and current from IGBT and diode is close to linear as in Fig. 3.28(a), a linear approximation of IGBT output voltage v_{ce} and current i_c, diode output voltage v_F and current i_F is obtained using

$$r_{SIGBT} = \frac{\Delta v_{ce}}{\Delta i_c}, \tag{3.172}$$

$$r_{Sd} = \frac{\Delta v_f}{\Delta i_f}. \tag{3.173}$$

In addition, the turn-off current is modeled using two linear slopes as shown in Fig. 3.28(b) [43]. The capacitor in each SM is discretized into a resistor r_{cap} in series with a historical voltage source v_{cap_h}(t-Δt). The r_1, v_1 and r_2, v_2 in Fig. 3.27(c) is decided by the gate signal

and arm current direction. In this way, each SM's Thevenin equivalent circuit in Fig.3.27(d) contains a time dependent inductance and historic voltage source [37], where r_{SM} and v_{SM} are given as

$$r_{SM} = \frac{r_2(r_1 + r_{cap})}{r_1 + r_2 + r_{cap}}, \tag{3.174}$$

$$v_{SM} = (\frac{v_2}{r_2} + \frac{v_1 + v_{cap\_h}(t - \Delta t)}{r_1 + r_{cap}})r_{SM}. \tag{3.175}$$

And the capacitor historic voltage is updated using the method of linear passive elements as following,

$$v_{cap\_h}(t - \Delta t) = v_{cap\_h}(t - 2\Delta t) + 2r_{cap}i_{cap}(t - \Delta t), \tag{3.176}$$

where

$$r_{cap} = \frac{2\Delta t}{C}. \tag{3.177}$$

This simplified model makes each arm of MMC containing n SMs in Fig. 3.23 replaced with a voltage source and a resistor as following,

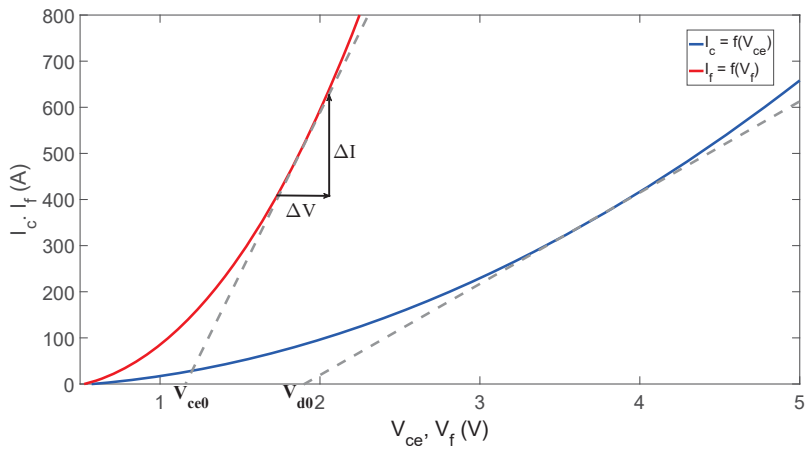
$$v_{arm} = \sum_{i=1}^{n} v_{SMi}, \tag{3.178}$$

$$r_{arm} = \sum_{i=1}^{n} r_{SMi}. \tag{3.179}$$

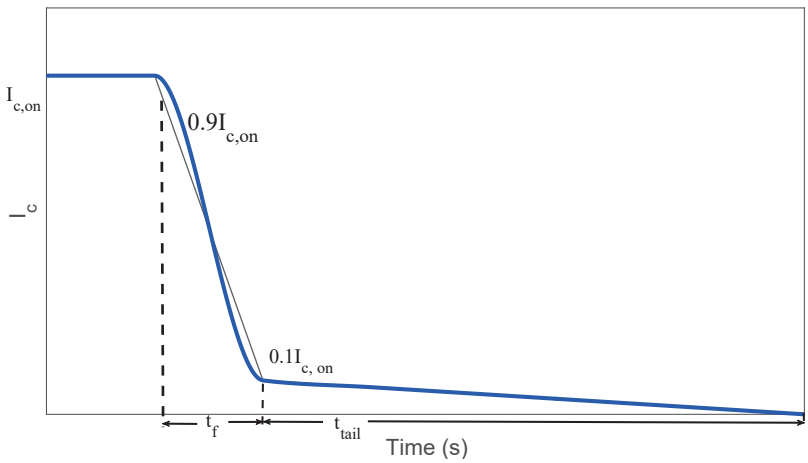
The arm current can be calculated using the arm equivalent model. Since each SM's input current is the same as arm current, the node voltage inside each SM can be updated in parallel using the solved arm current for the current time point.

3.8.2 Parallel Massive-thread Mapping

As shown in Fig. 3.28, only one kernel containing 6n threads is involved to calculate solve the MMC circuit using behavior-based models. Assuming a system containing n SMs in each arm, there are 6n SMs in total for 3-phase. According to gate signal and previous time arm current direction, v_1, r_1 and v_2, r_2 in each SM are obtained to build the equivalent Thevenin circuit using (3.174) and (3.175). Using the sum of n SMs in each arm, the arm current are solved for node voltage update in each SM. Based on the fact that once the arm current is solved, the SM only contains 2 node voltages to be updated, only one thread is needed for each SM.



(a) Linear Approximation of IGBT and diode



(b) IGBT measurement model for turn-off current

Algorithm 8 Behavior-based MMC solution kernel

```
while t < t_end do
    Update v1, r1 and v2, r2 in each SM
    Calculate the v_SM(t), r_SM(t) in each SM as (3.174) and (3.175)
    Sum the n SMs in each arm and solve i_arm(t)
    Node voltage and current solution for each SM
    t ← t + Δt
```

▷ $Kernel_0$

3.9 Summary

In this chapter, massively parallel implementation models of electronic components including physics-based IGBT and power diode, numerical nonlinear equation solver using Newtone-Raphson method, linear equation solver including GEMES and LUDMES. Based on MMC Jacobian matrix characteristic, a blocked Jacobian matrix equation solver is developed using partial LU decomposition improving parallelism. To increase simula-

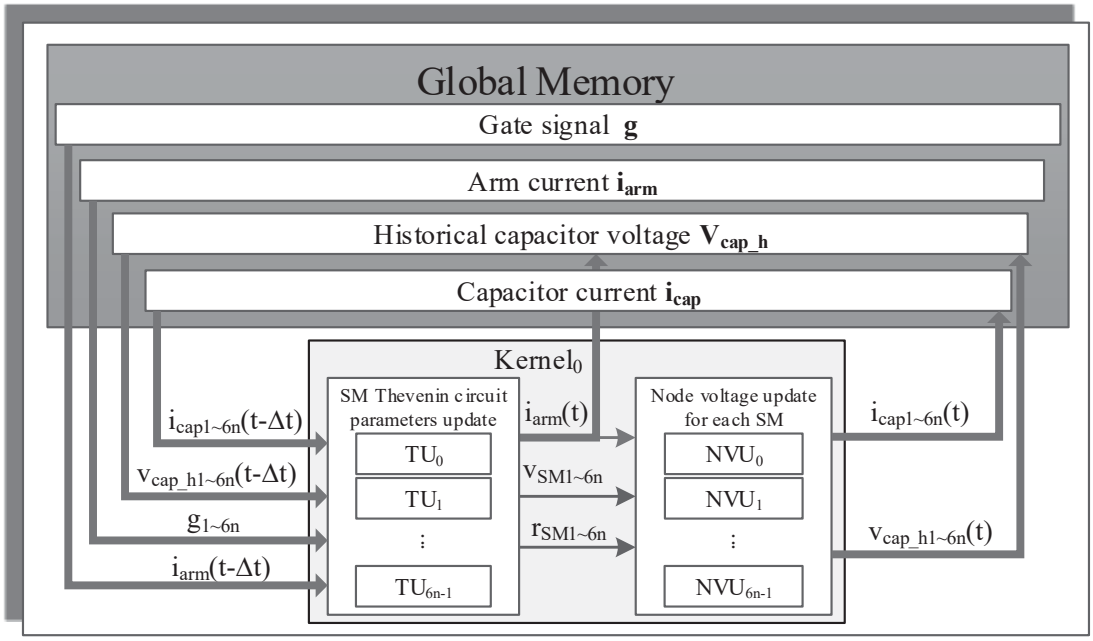


Figure 3.28: Massive thread parallel implementation of behavior-based MMC solution

tion efficiency, a PCVTSM is developed and massively implemented. Using the detailed physics-based model increases calculation complexity while bringing accurate details. In addition, to achieve higher level of MMC circuit, the behavior-based SM model is implemented in massive-thread architecture. In system level, the control strategy of MMC circuit including active and reactive power control and averaging and balancing control are explained. With the methods mentioned above, massively parallel implementation for power electronic circuit is practically and applicable.

4
MMC Case Study and Data Analysis

Based on the physics-based device-level, as well as linear behavioral, IGBT and power diode models and massively parallel implementations in Chapter 3, the large-scale MMC case is tested for results and simulation execution time comparison between massively parallel simulation and sequential simulation. Both single-phase and three-phase MMC circuits are simulated. Single-phase results are compared with SaberRD® and three-phase results are compared with C++ sequential simulation result.

4.1 Test case for 5-level Physics-based MMC Simulation Case Study

In Chapter 3, the structure and modulation strategy of the MMC were explained. The case of a single-phase 5-level MMC circuit is simulated both on GPU and SaberRD®. The converter has 8 SMs both in upper-arm and lower-arm. Tab. 4.1 lists the hardware and

Table 4.1: Environment Specification

	GPU		CPU
	NVIDIA Kepler™ GK110 GeForce GTX Titan Black	NVIDIA Pascal™ GP104 GeForce GTX 1080	Intel Ivy Bridge Core™ i7-3770
Cores	2880	2560	4 (8 threads)
Frequency	889 MHz	1607 MHz	3.4 GHz
Memory	6 GB	8GB	8 GB

Table 4.2: 5-level Single-phase MMC Circuit Specification

System specification					
Arm inductance L_a	5 mH	Load inductance L_L	5 mH		
Load Resistance R_L	4.6 Ω	SM capacitance C_m	4 mF		
DC voltage V_{dc}	1000 V	Gate resistance	100Ω		
System frequency f_s	60 Hz	Carrier signal frequency f_c	2500 Hz		
Simulation time t_s	100 ms	Initial time-step size h	1 ns		
IGBT specified parameters					
A	0.1 cm^2	AGD	0.05 cm^2	I_{sne}	6.5e-14 A
NB	2.0×10^{14} cm^{-3}	V_{crit}	0.6 V	τ_{HL}	0.6 s
K_f	1.0	K_p	0.38 A/V	θ	0.02 V^{-1}
V_t	4.7 V	W_B	0.009 cm	BVf	1
BVn	4	C_{gs}	6.2×10^{-10} F	C_{oxd}	1.75×10^{-9} F
LTD	1×10^{-3} V				
Diode specified parameters					
I_S	10^{-14} A	τ	5 μs	T_M	5 μs
V_T	0.0259 V	m	0.5	C_{j0}	1nF
V_J	0.7 V	I_{SE}	10^{-22}	R_c	10^{-3}

software specification of the GPU and CPU used in the work. The parameters of physics-based device level IGBT and power diode models adopted in the case study are listed in Tab. 4.2.

$$G^{5-level}_{79\times79}\Delta V_{79\times1} = -I_{79\times1}. \tag{4.1}$$

As illustrated in Chapter 3, the Jacobian matrix to solve a single IGBT is a 5×5 matrix G^{IGBT} in (3.90), adding an anti-parallel diode brings one more circuit node inside the diode. Thus, the Jacobian matrix for the pair is a 6×6 matrix. For the SM consisting of 2 pairs of IGBT and diode, the Jacobian matrix G^{SM} has the dimension of 11×11. In the 5-level MMC circuit containing 8 cascading SMs, there are 79 node voltages, except the two nodes connecting to DC voltage source, to be calculated. To solve the following Newton-Raphson iterative equations not only takes lot of execution time, but also brings convergence problems.

Another challenge of solving the high level MMC circuit is the ill conditioning of the

Jacobian matrix. The condition number of Jacobian matrix influences the solution accuracy. The condition of a square nonsingular matrix A is defined as following:

$$cond(A) = ||A|| \cdot ||A^{-1}||. \tag{4.2}$$

If the condition number is close to 1, it means the matrix is well-conditioned, whereas if the condition number is too large, the matrix is deemed as ill-conditioned. In equation (4.1), if $cond(G^{5-level}_{79\times79})$ is too large, a small error in -$I_{79\times1}$ will cause a large inaccuracy in the iterative solution. Given the example of the single IGBT Jacobian matrix during simulation, the matrix has the following values. $G^{IGBT} =$

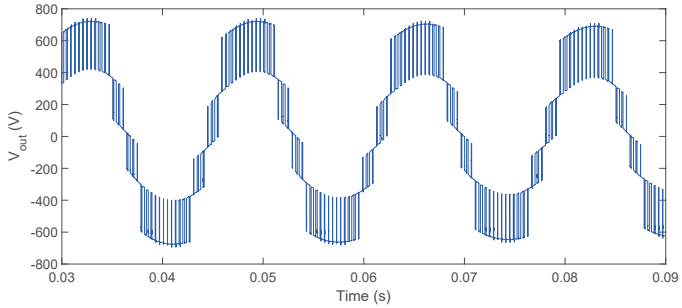
$$\begin{bmatrix} 1.17722\times10^{-19} & -1.2410\times10^{-9} & -1.0000\times10^{-12} & -5.3015\times10^{-10} & 0 \\ -1.2410\times10^{-9} & 3.9958\times10^{-9} & 0 & -2.7558\times10^{-9} & 0 \\ 0 & 0 & 0.5454 & 0 & -0.5454 \\ -5.3015\times10^{-10} & -2.7548\times10^{-9} & 0 & 4.3443\times10^{-9} & -1.0593\times10^{-9} \\ -2\times10^{-12} & 0 & -0.5454 & -1.0583\times10^{-9} & 0.5454 \end{bmatrix} \tag{4.3}$$

Tab. 4.3 lists the condition number of the G^{IGBT} during a iterative equation solution process taking 5 iterations to converge. The condition numbers of the Jacobian matrix are quite large which means the whole system is very sensitive to errors and it is hard to find accurate solution. Due to these characteristic, SaberRD® cannot solve single-phase MMC systems containing more than 8 SMs, there is no existing DC solution and the iterative process fails.

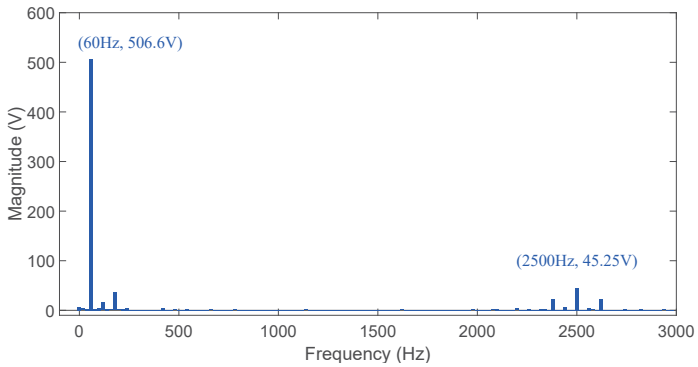
The system-level simulation results comparison between SaberRD® and the GPU simulation is shown in Fig. 4.1(a) to Fig. 4.5(b). The result proves the GPU simulation accuracy. In addition, SaberRD® is more sensitive to the system parameters than the GPU code. It will not give complete simulation result when using default setting values. Adjusting the local truncation error (LTE) limit and target Newton-Raphson iteration number will help to complete the simulation. The use of a variable time stepping method for a system having a large condition number, means that the simulation is sensitive to small errors, and the convergence problem is influenced by several factors. Changing the LTE

Table 4.3: Condition number of iterative Jacobian matrix G^{IGBT}

Number of iteration	1	2	3	4	5
cond(G^{IGBT})	2.3813×10^{17}	1.6154×10^{17}	5.7653×10^{16}	9.2175×10^{16}	7.1517×10^{17}



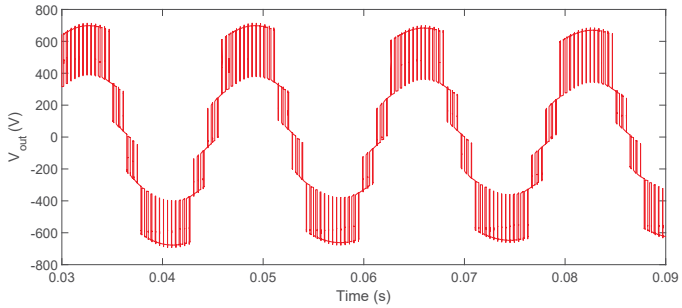
(a) 5-level MMC circuit output voltage SaberRD® simulation result



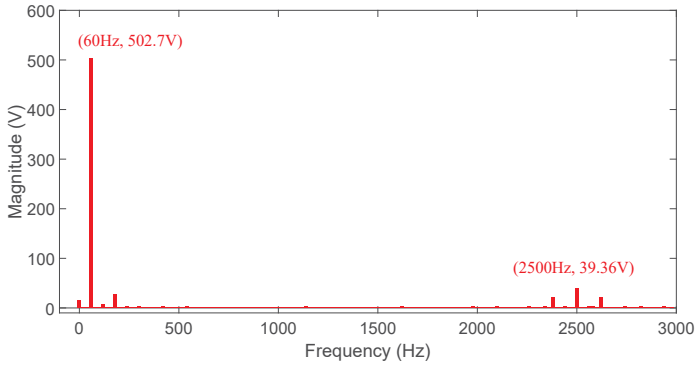
(b) FFT analysis of 5-level MMC circuit output voltage from SaberRD® simulation

Figure 4.1: 5-level MMC circuit output voltage waveform and FFT analysis from SaberRD®

limit affects the sampling time points, which in some case makes the simulator skip some points that do not converge. While in the GPU code, the convergence problem also exists; nevertheless, instead of solving a 79×79 matrix equation directly as in SaberRD®, solving a 5×5 block matrix equation using partial LU decomposition reduces the chances of non-convergence. Fig .4.1(b), Fig. 4.2(b), and Fig. 4.4(b), Fig. 4.5(b) show the output voltage and load current Fast Fourier transform (FFT) analysis of SaberRD® and the GPU simulation results. Since the voltage and current result are obtained using variable time-step method, to obtain FFT result, a sampling rate of 20480 Hz is performed in advance. Linear interpolation, although it is easy to implement, it is prone to distortion, is adopted to produce samples at fixed rate. Beside the base frequency of 60 Hz, the harmonic frequency around f_c of 2.5 kHz is obvious. Due to the inductors in the circuit, the load current has lower harmonics than voltage. And the capacitor voltage comparison between SaberRD® and GPU is shown in Fig. 4.3(a) and Fig. 4.3(b). With averaging and balancing control, the capacitor voltages are kept around $2V_{dc}/n_{arm}$, where n_{arm} refers to the number of SMs per arm.

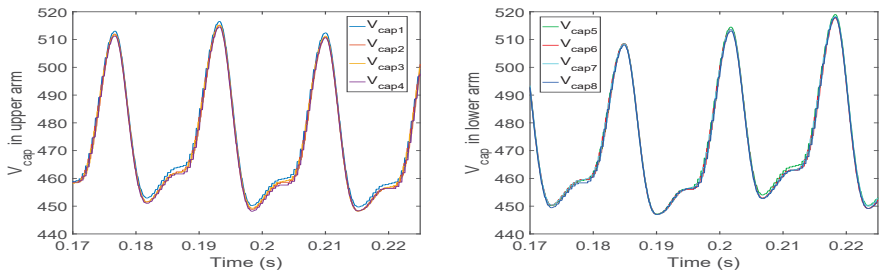


(a) 5-level MMC circuit output voltage the GPU simulation result

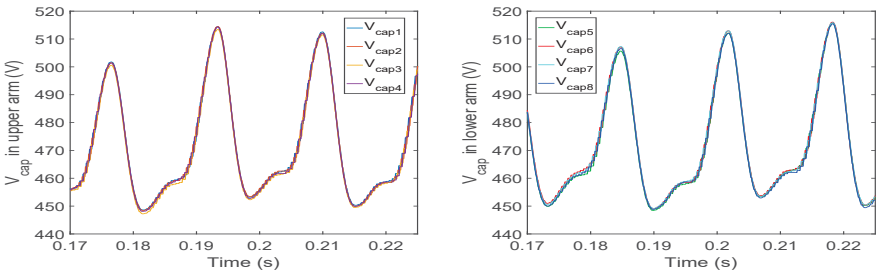


(b) FFT analysis of 5-level MMC circuit output voltage from the GPU simulation

Figure 4.2: 5-level MMC circuit output voltage waveform and FFT analysis from GPU simulation

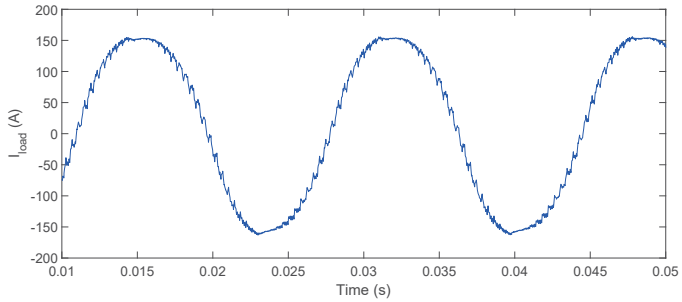


(a) Capacitor voltages from SaberRD®

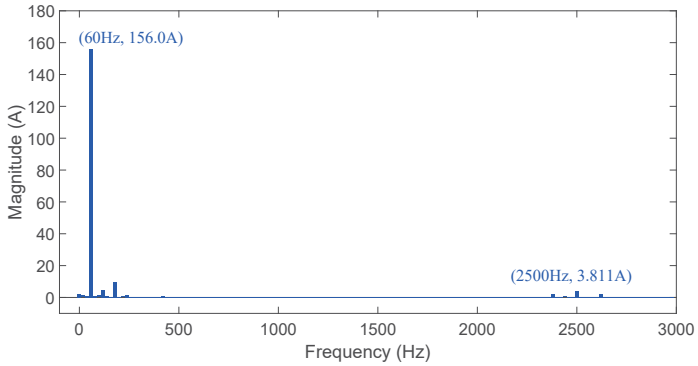


(b) Capacitor voltages from the GPU simulation

Figure 4.3: Capacitor voltages in upper and lower arm form SaberRD® and GPU

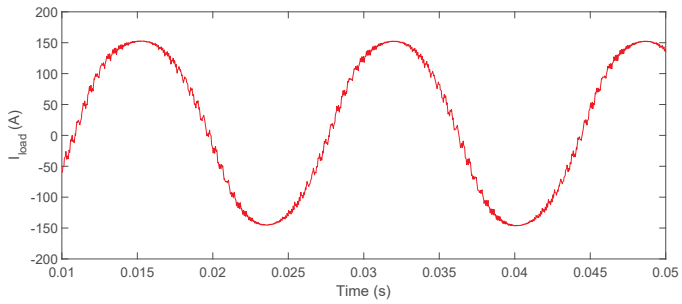


(a) 5-level MMC circuit load current SaberRD® simulation result

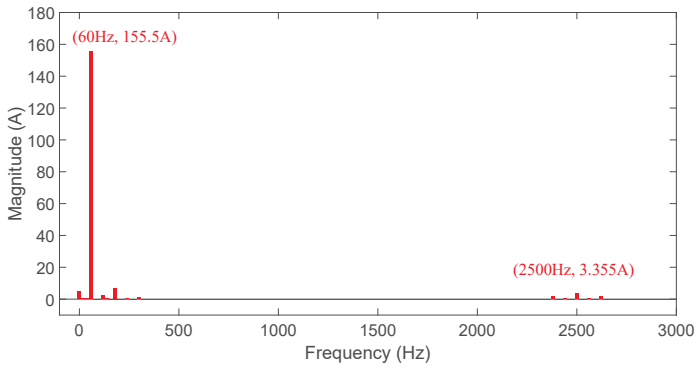


(b) FFT analysis of 5-level MMC circuit load current from SaberRD® simulation

Figure 4.4: 5-level MMC circuit load current waveform and FFT analysis from SaberRD®

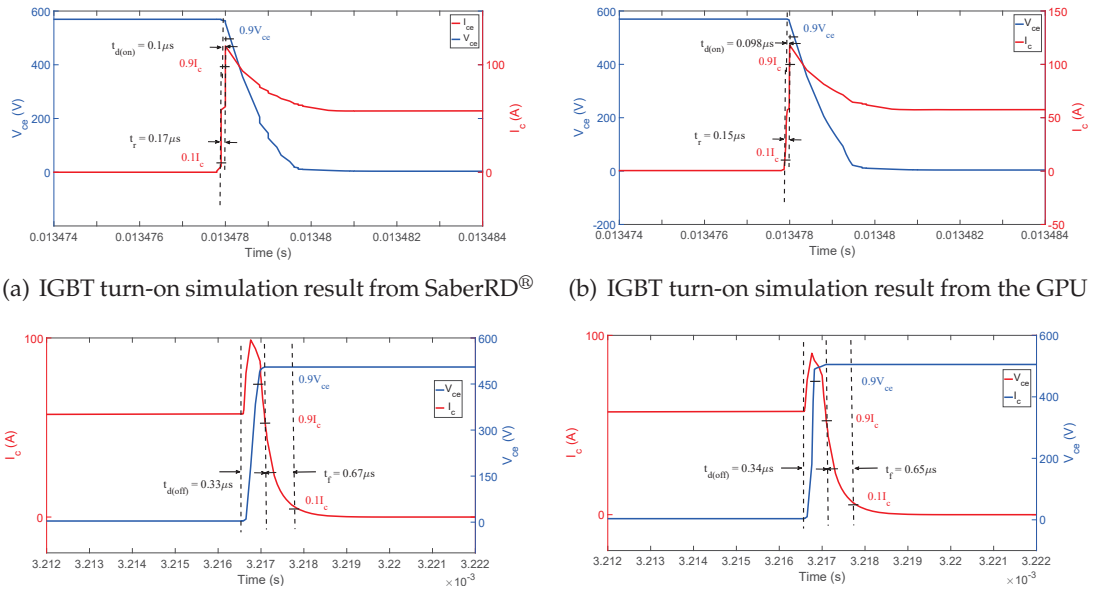


(a) 5-level MMC circuit load current the GPU simulation result



(b) FFT analysis of 5-level MMC circuit load current from the GPU simulation

Figure 4.5: 5-level MMC circuit load current waveform and FFT analysis from GPU



(a) IGBT turn-on simulation result from SaberRD®

(b) IGBT turn-on simulation result from the GPU

(c) IGBT turn-off simulation result from SaberRD®

(d) IGBT turn-off simulation result from the GPU

Figure 4.6: IGBT turn-on turn-off voltage and current waveform from SaberRD® and the GPU simulation

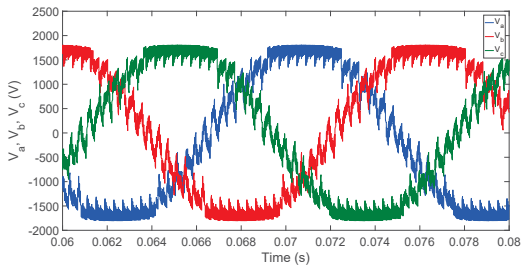
The device-level simulation result is given in Fig. 4.1 which shows the detailed turn-on and turn-off time, voltage and current. Tab. 4.4 gives the power dissipation of S_2 and D_2 in during switching and conducting situation. The power loss in IGBT is calculated using the following equation

$$P^{IGBT} = \frac{\int v_{ce}(t) i_c(t) dt}{T_s}, P^{diode} = \frac{\int v_f(t) i_f(t) dt}{T_s}, \tag{4.4}$$

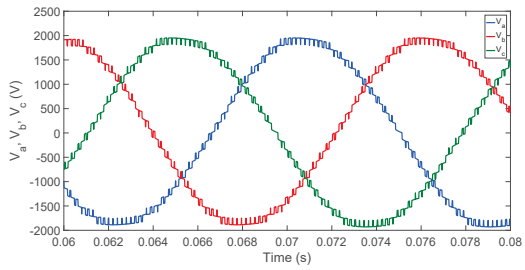
where v_f is the voltage cross the diode and i_f stands for the current through current, and T_s is the switching period.

Table 4.4: Device-level switching time and power dissipation for S_2 and D_2

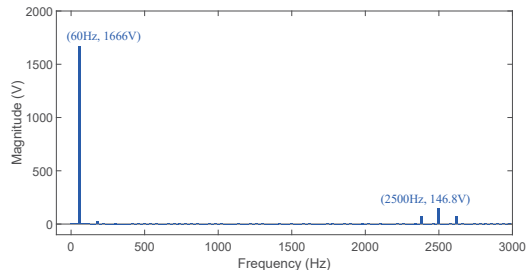
Switching time (μs)			Power dissipation (W)		
	Saber®	GPU		Saber®	GPU
$t_{d(on)}^{IGBT}$	0.10	0.098	P_{on}^{IGBT}	112.31	113.02
t_r^{IGBT}	0.17	0.15	P_{off}^{IGBT}	75.01	74.84
$t_{d(off)}^{IGBT}$	0.33	0.34	P_{cond}^{IGBT}	287.52	289.06
t_f^{IGBT}	0.67	0.65	P_{cond}^{Diode}	7.48	7.55
t_{rr}^{Diode}	0.66	0.64	P_{rr}^{Diode}	9.95	10.07



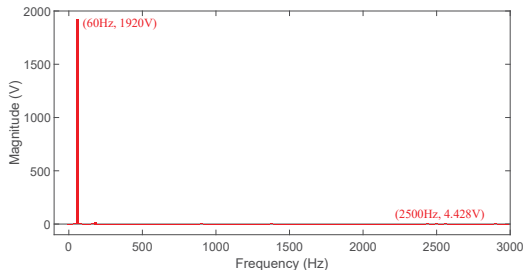
(a) 3-phase 11-level physics-based MMC circuit output voltage from GPU simulation



(b) 3-phase 11-level behavior-based MMC circuit output voltage from GPU simulation



(c) FFT analysis of V_a in 3-phase 11-level physics-based MMC circuit from the GPU simulation



(d) FFT analysis of V_a in 3-phase 11-level behavior-based MMC circuit from the GPU simulation

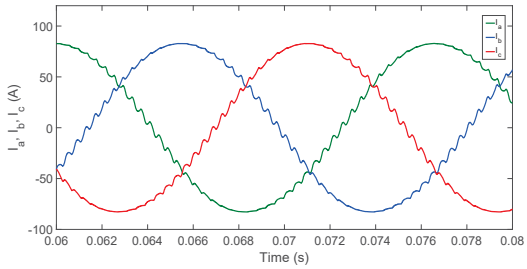
Figure 4.7: 3-phase 11-level MMC circuit output voltage waveform and FFT analysis using physics-based and behavior-based model from the GPU simulation

4.2 Test Case for 3-phase 11-level Physics-based MMC Simulation

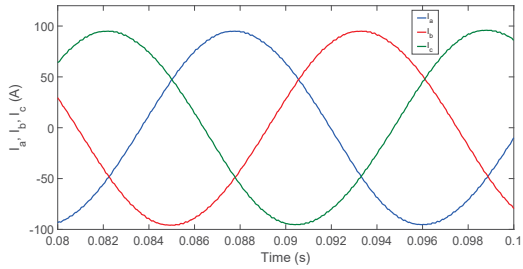
The 3-phase MMC circuit simulation results are compared between GPU and CPU codes since SaberRD® can only converge and give result for the 3-phase 2 SM per phase with time consumption of 183 seconds. The 3-phase 11-level MMC system parameters are listed

Table 4.5: 3-phase 11-level MMC Circuit Specification

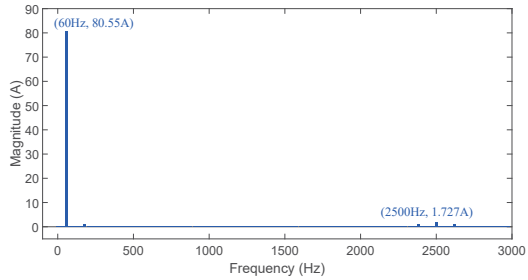
System specification			
Arm inductance L_a	5 mH	Load inductance L_L	5 mH
Load Resistance R_L	20 Ω	SM capacitance C_m	4 mF
DC voltage V_{dc}	2000 V	Gate resistance	100Ω
AC voltage source V_{sl_l}	2000 V	Rated power P_{rated}	600 kW
System frequency f_s	60 Hz	Carrier signal frequency f_c	2500 Hz
Simulation time t_s	100 ms	Initial time-step size h	1 ns



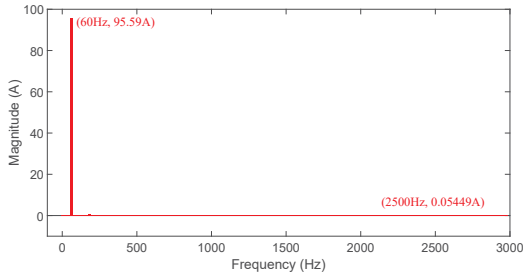
(a) 3-phase 11-level physics-based MMC circuit load current from the GPU simulation



(b) 3-phase 11-level behavior-based MMC circuit load current from the GPU simulation



(c) FFT analysis of I_a in 3-phase 11-level physics-based MMC circuit from the GPU simulation

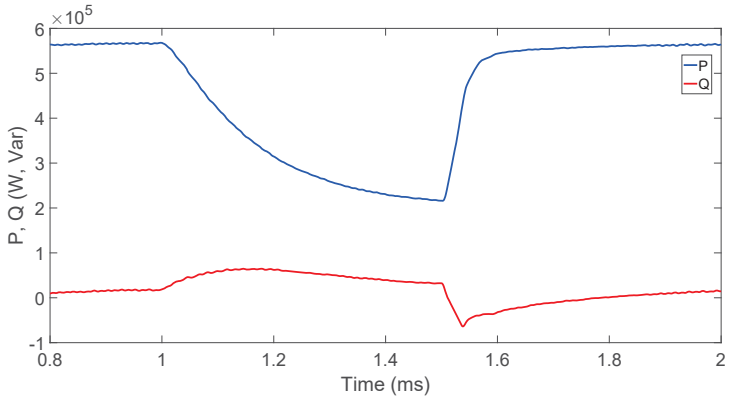


(d) FFT analysis of I_a in 3-phase 11-level behavior-based MMC circuit from the GPU simulation

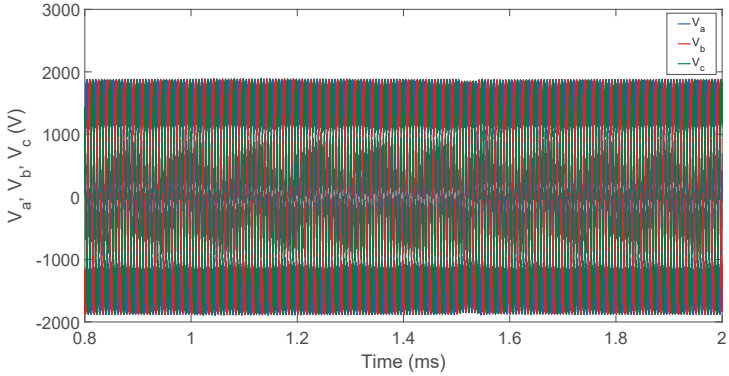
Figure 4.8: 3-phase 11-level MMC circuit load current waveform and FFT analysis using physics-based and behavior-based model from the GPU simulation

in Tab. 4.5 with the same IGBT and diode parameters as in the single-phase case. Fig. 4.7(a) and Fig. 4.8(a) give the output voltage and load current of each phase simulated on the GPU. The FFT analysis of phase a voltage and current are given in Fig. 4.7(c) and Fig. 4.8(c) with a sampling frequency of 20480 Hz. Compared with Fig. 4.2(b) and 4.5(b), the harmonics are smaller in both voltage and current waveforms due to increasing levels. For comparison, the 3-phase 11-level MMC circuit using behavior-based model is also simulated using the same system specification. As shown in Fig. 4.7(b) and Fig. 4.8(b), both the voltage and current waveforms using behavior-based model are smoother than the ones using physics-based models. The FFT analysis of voltage and current comparison is more obvious in Fig. 4.7(c), Fig. 4.7(d), Fig. 4.8(c) and Fig. 4.8(d). Due to the nonlinear capacitance and dynamic current sources in physics-based models, it is more sensitive to transient in each nonlinear device.

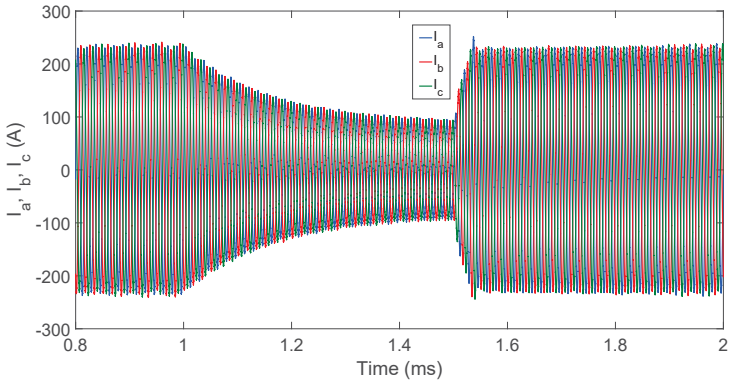
In Fig. 4.9, the active power control results are given by changing the reference active power value from 0.9 pu to 0.3 pu during 1 ms to 1.5 ms. As shown in Fig. 4.9(b), the output voltage almost keep steady during the change of active power reference, which verify the averaging and balancing capacitor voltage control. And in Fig. 4.9(c), the amplitude of 3-phase current varies to make output active power to follow the reference value, proving



(a)



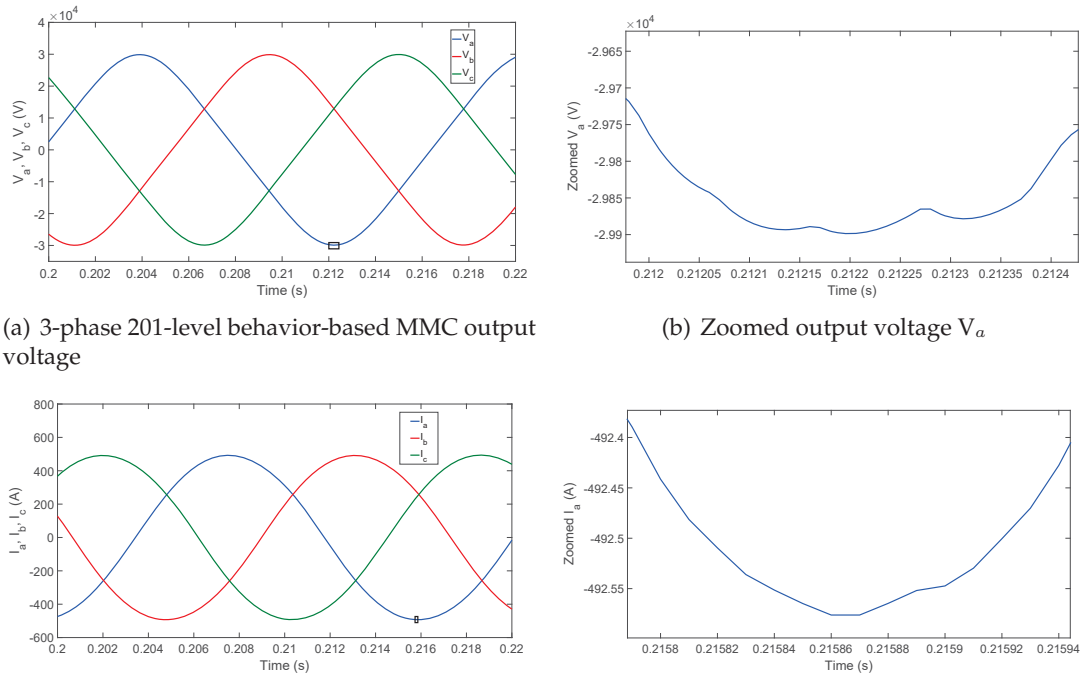
(b)



(c)

Figure 4.9: 3-phase 11-level MMC circuit active power control results

the power control strategy in Chapter 3. The CPU simulation gives exactly the same output voltage and current except with a higher time consumption.



(a) 3-phase 201-level behavior-based MMC output voltage

(b) Zoomed output voltage V_a

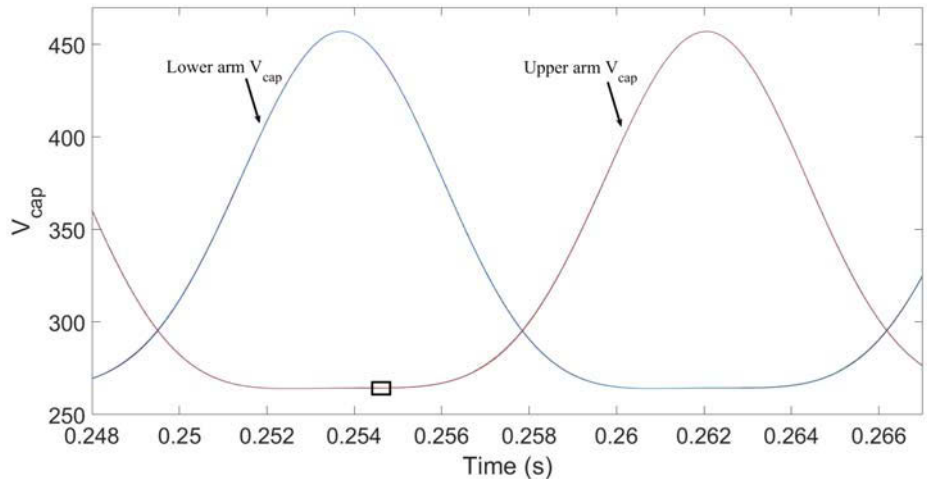
(c) 3-phase 201-level MMC load current

(d) 3-phase 201-level MMC load current

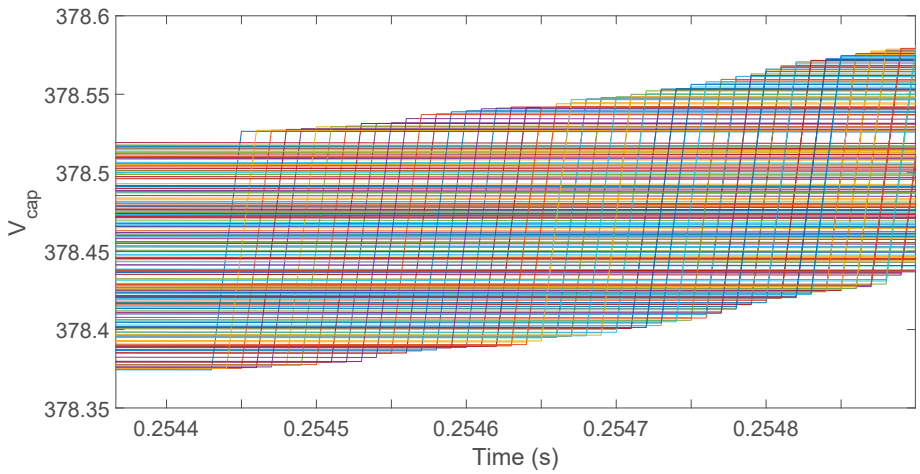
Figure 4.10: 3-phase 201-level behavior-based MMC output voltage and load current

4.3 3-phase 201-level Behavior-based MMC Simulation

A behavior-based MMC circuit is simulated for comparison between GPU and CPU for large-scale power electronic converter application. The converter contains 400 SMs on each phase having the specification in Tab. 4.6. Fixed time-step scheme is adopted for execution time comparison since the time-step size is much higher than that of physics-based model case. The output voltage and load current simulation output is shown in Fig. 4.10(a) and Fig. 4.10(c), which is highly close to sinusoidal compared with lower level result. From the zoomed waveform of voltage and current in Fig 4.10(b) and Fig. 4.10(d), there are still harmonic component in the waveforms, similarly, current waveform is more close to sinusoidal waveform due to the arm and load inductance. Each capacitor voltage in Fig. 4.11(a) is around reference capacitor voltage 380V ($2V_{dc}/n_{arm}$) with capacitor averaging and balancing control strategy. Fig. 4.11(b) gives the zoomed waveform of the rectangular in Fig. 4.11(a) showing the detail of capacitor voltage change in the same arm.



(a) Capacitor voltages



(b) Zoomed capacitor voltages in upper arm

Figure 4.11: 3-phase 201-level capacitor voltages in upper and lower arm

Table 4.6: 3-phase 201-level MMC Circuit Specification

System specification			
Arm inductance L_a	150 mH	Load inductance L_L	3 mH
Load Resistance R_L	5 Ω	SM capacitance C_m	4 mF
DC voltage V_{dc}	38 kV	Gate resistance	100Ω
System frequency f_s	60 Hz	Carrier signal frequency f_c	2500 Hz
Simulation time t_s	0.5 s	Fixed time-step size h	10 μs

4.4 Execution Time Comparison

The single-phase MMC circuits are tested from 2-level to 11-level as shown in Tab. 4.7 for 100 ms time simulation using variable time-step method. Since SaberRD® is hard to converge when containing more than 8 SMs, the execution time comparison for higher levels is between GPU and CPU codes. The CPU code has similar execution time with SaberRD® based on the completed cases. Both CPU code and SaberRD® are slower than GPU code when there are more than 8 SMs as shown in Fig. 4.12. In addition, the GPU code is running on both GK110 and GP104 to get execution time T_{GPU1}, T_{GPU2} and speed-up SU_1, SU_1 compared with CPU code. With the development of GPUs, GP104 achieves higher speed-up compared with GK110. The 3-phase execution time comparison between GPU and CPU codes is listed in Tab. 4.8. In the 3-phase MMC case, the computation of the three phases is naturally paralleled for the GPU; therefore, as shown in Fig. 4.13, the advantage of GPU computation and speed-up is more obvious.

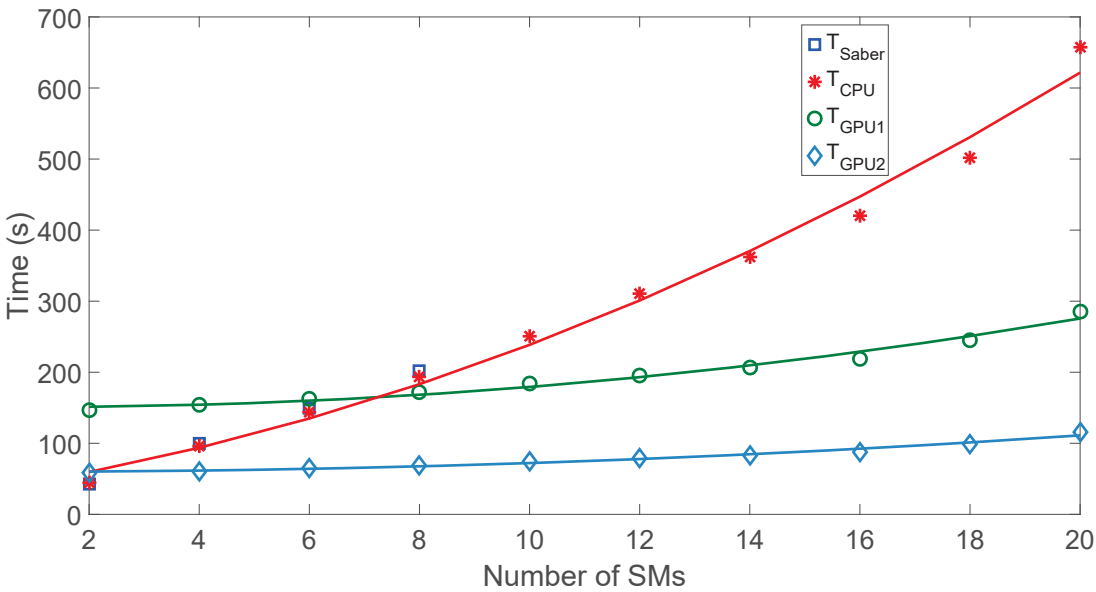


Figure 4.12: Single-phase physics-based MMC circuit execution time comparison

Table 4.7: SaberRD®, CPU and GPU execution time of single-phase physics-based MMC

n_{SM}	n_{IGBT}	n_{Vlevel}	T_{Saber} (s)	T_{CPU}(s)	T_{GPU1} (s)	SU_1	T_{GPU2} (s)	SU_2
2	4	2	43	44.8	147.1	0.30	58.9	0.76
4	8	3	100	95.9	154.4	0.62	60.7	1.63
6	12	4	149	143.8	162.5	0.88	65.4	2.20
8	16	5	202	193.5	171.7	1.13	69.2	2.82
10	20	6	-	250.9	184.2	1.36	74.1	3.39
12	24	7	-	310.3	195.7	1.59	79.3	3.91
14	28	8	-	361.8	206.5	1.75	83.2	4.35
16	32	9	-	420.5	218.6	1.92	87.9	4.78
18	36	10	-	501.5	245.2	2.08	98.7	5.08
20	40	11	-	657.8	285.5	2.30	115.4	5.70

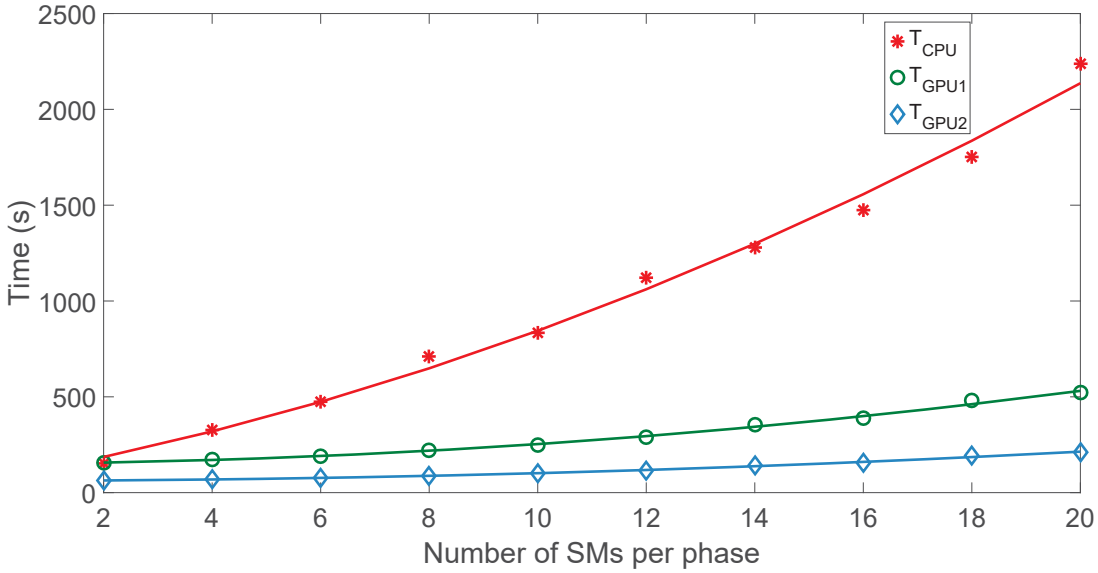


Figure 4.13: 3-phase physics-based MMC circuit execution time comparison

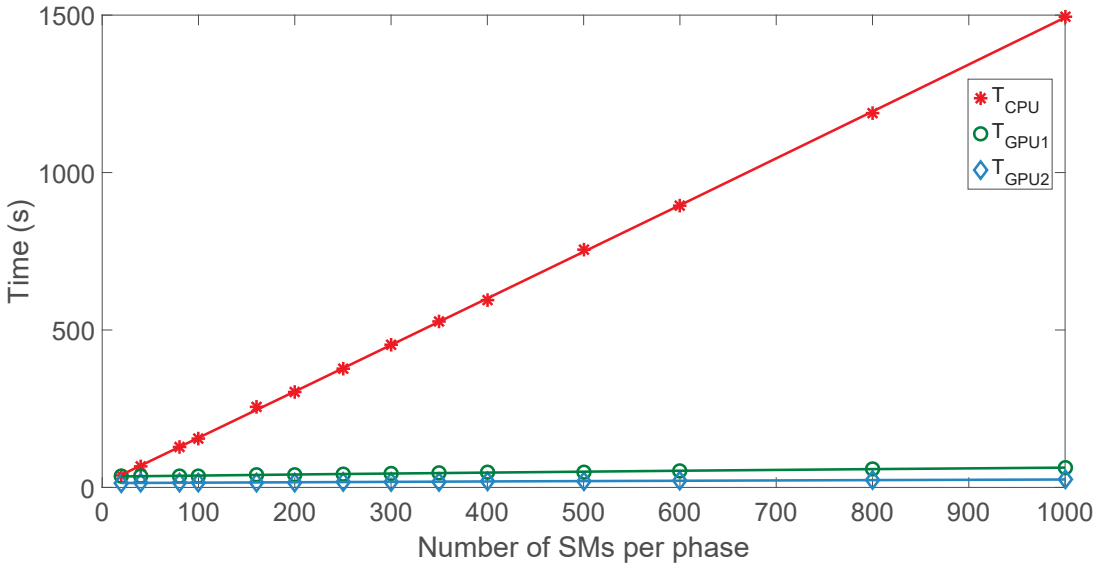
Table 4.8: CPU and GPU simulation execution time of 3-phase physics-based MMC

n_{SM} per phase	n_{IGBT}	$n_{V_{ln}}$	$n_{V_{ll}}$	T_{CPU}(s)	T_{GPU1} (s)	SU_1	T_{GPU2}	SU_2
2	12	2	3	150.8	156.2	0.97	63.1	2.39
4	24	3	5	327.9	172.9	1.90	70.0	4.68
6	36	4	7	473.5	191.7	2.47	76.9	6.16
8	48	5	9	710.3	219.2	3.24	88.2	8.05
10	60	6	11	834.1	248.5	3.36	99.4	8.39
12	72	7	13	1121.1	289.2	3.88	115.2	9.73
14	84	8	15	1279.2	354.3	3.61	142.3	8.99
16	96	9	17	1475.4	387.9	3.80	156.5	9.43
18	108	10	19	1752.4	480.2	3.65	193.1	9.08
20	120	11	21	2237.3	521.1	4.29	209.6	10.67

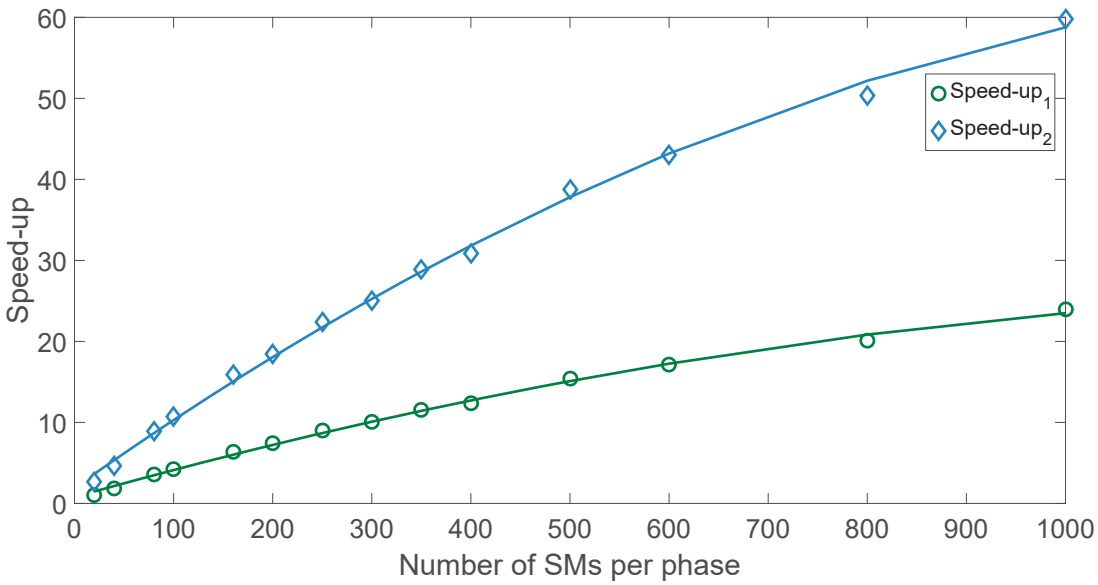
The behavior-based 3-phase MMC circuits are tested from 11-level to 201-level for 0.5 s time simulation using 10 μs fixed time-step method as shown in Fig. 4.14(a) and Tab. 4.9. Fig. 4.15 and Fig. 4.16 gice the output voltage and load current waveforms from 21-level and 501-level in each phase of 3-phase MMC circuit, as well as the capacitor voltage in upper and lower arm, with the growth of levels, the arm inductance increases to give smoother voltage and current waveforms. The simplified model of SM makes the whole arm of the MMC can be treated as a equivalent Thevenin circuit, therefore, the calculated arm current makes the computation for each SM independent from each other. For the behavior-based model of SM, there are two node voltage in the SM, the SM output voltage and capacitor voltage need to be updated. With the growth of SM numbers in each phase, the parallel update makes the speed-up of GPU simulation significant.

Table 4.9: CPU and GPU simulation execution time of behavior-based 3-phase MMC

n_{SM} per phase	n_{IGBT}	$n_{V_{ln}}$	$n_{V_{ll}}$	T_{CPU}(s)	T_{GPU1} (s)	SU_1	T_{GPU2} (s)	SU_2
20	120	11	21	37.2	35.5	1.05	14.1	2.64
40	260	21	41	67.4	35.9	1.88	14.4	4.68
80	480	41	81	129.2	36.2	3.57	14.5	8.91
100	600	51	101	156	36.6	4.26	14.6	10.68
160	960	81	161	255.8	40.2	6.36	16.1	15.89
200	1200	101	201	302.8	40.9	7.40	16.4	18.46
250	1500	126	251	376.2	41.9	8.98	16.8	22.39
300	1800	151	301	453.2	45.1	10.05	18.1	25.04
350	2100	176	351	526.2	45.6	11.54	18.2	28.91
400	2400	201	401	595.6	48.3	12.33	19.3	30.86
500	3000	251	501	755.2	48.9	15.44	19.5	38.73
600	3600	301	601	894.5	52.2	17.14	20.8	43
800	4800	401	801	1188.2	59.1	20.10	23.6	50.34
1000	6000	501	1001	1495.6	62.5	23.93	25.0	59.8

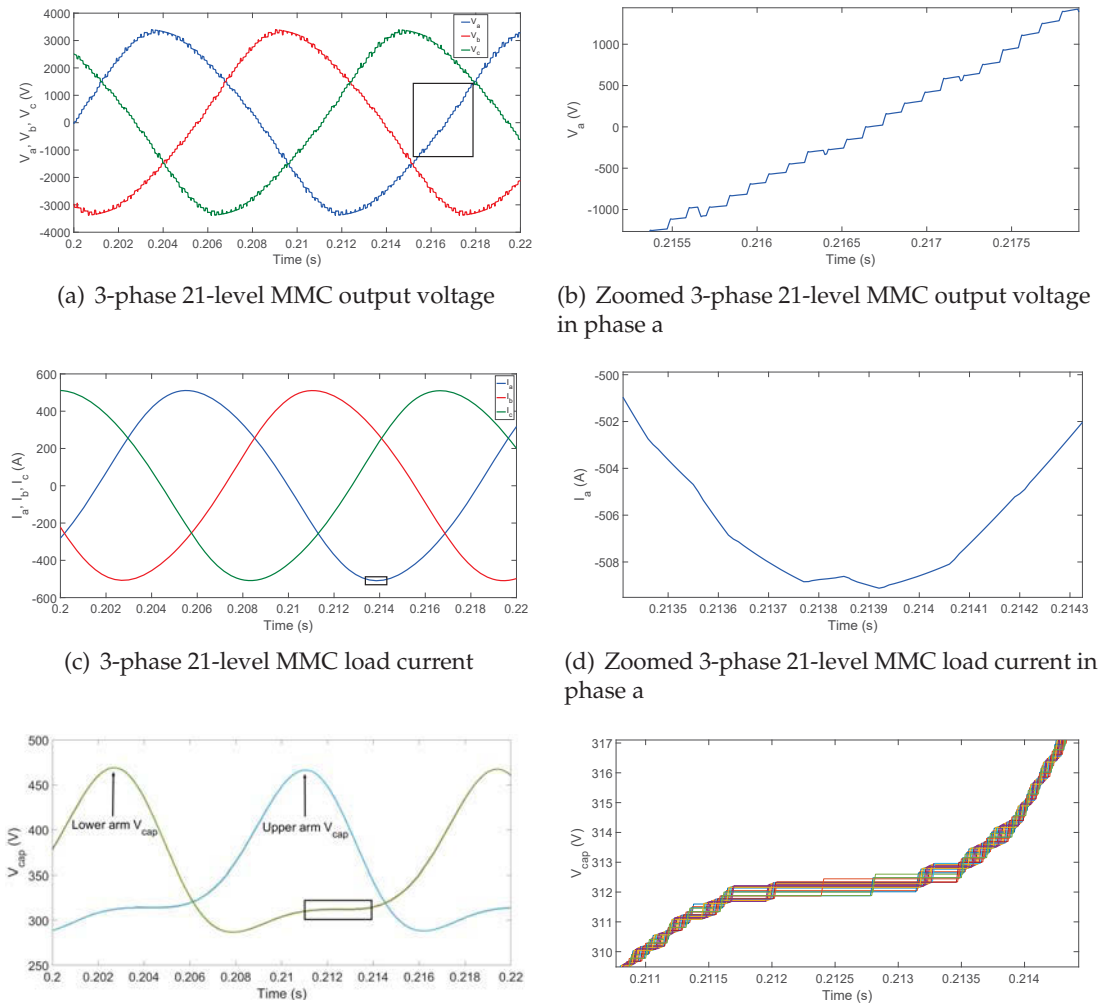


(a) 3-phase behavior-based MMC circuit execution time comparison of CPU and GPU



(b) Speed-up comparison of GTX GK110 and GP104

Figure 4.14: 3-phase behavior-based MMC circuit execution time and speed-up comparison



(a) 3-phase 21-level MMC output voltage

(b) Zoomed 3-phase 21-level MMC output voltage in phase a

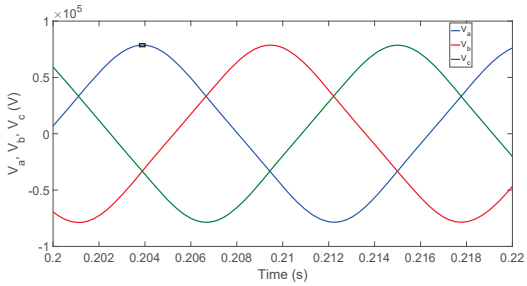
(c) 3-phase 21-level MMC load current

(d) Zoomed 3-phase 21-level MMC load current in phase a

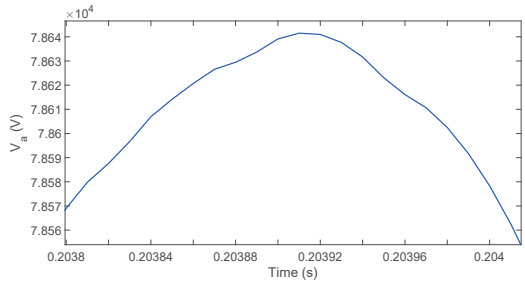
(e) 3-phase 21-level MMC capacitor voltage in upper and lower arm

(f) Zoomed 3-phase 21-level MMC capacitor voltage in lower arm

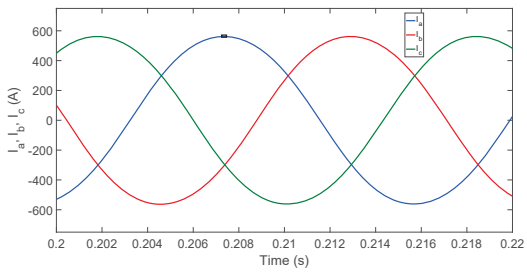
Figure 4.15: 3-phase 21-level behavior-based MMC circuit simulation results



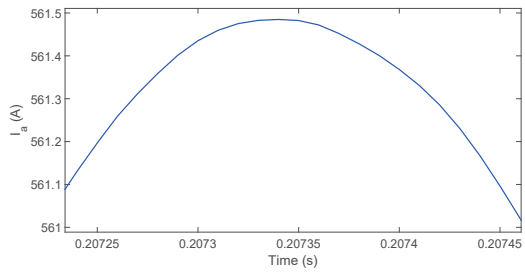
(a) 3-phase 501-level MMC output voltage



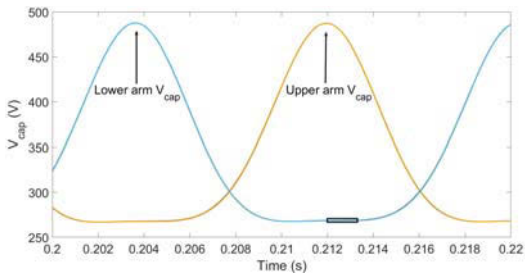
(b) Zoomed 3-phase 501-level MMC output voltage in phase a



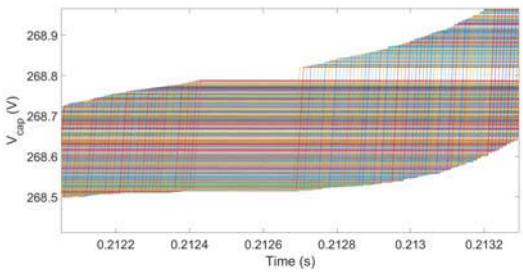
(c) 3-phase 501-level MMC load current



(d) Zoomed 3-phase 501-level MMC load current in phase a



(e) 3-phase 501-level MMC capacitance voltage in upper and lower arm



(f) Zoomed 3-phase 501-level MMC capacitor voltage in lower arm

Figure 4.16: 3-phase 501-level behavior-based MMC circuit simulation results

4.5 Summary

In this chapter, the cases of single-phase 5-level and 3-phase 11-level MMC circuits are simulated to verify the physics-based IGBT and power diode model in a practical application. And the case of 201-level 3-phase MMC circuit using linear behavior-based models is tested for large-scale massively parallel simulation application. The massively parallel implementation of MMC circuit is advantageous and gains speed-up compared with serial CPU programming for both physics-based and linear behavior-based cases. In addition, with the growth of converter levels, the speed-up grows significantly, especially for 3-phase cases. However, the main limitation of higher level MMC configurations simulation using physics-based models is the convergence problem due to the high nonlinearity of the IGBT and power diode models. Since the system Jacobian matrix is ill-conditioned, the accuracy and convergence are directly influenced. The Jacobian matrix size increases by the factor of n^2, with n being the SM number in single-phase. With the partial block Jacobian matrix decomposition, the Jacobian matrix size being dealt within each block is constant. Therefore, the total time consumption of GPU simulation increases much more slowly than that for the CPU when the used memory size is within limitation. When dealing with the simplified behavior-based IGBT and diode models inside each SM, the parallelism is based on the ability to calculate the arm current, which is also the output current of each SM in the same arm. Since the reduced size of Jacobian matrix compared with that of physics-based models, the ability to calculate arm current makes the Jacobian matrix naturally decomposed to each SM, which is suitable for massively parallel computation.

5 Conclusion and Future Work

System-level and device-level simulation provides different focus in power electronic converter simulation. Adopting physics-based device-level models to build a multiple level power electronic converters gives a chance to give insight into the device while solving the entire system. The cost is to accomplish the high computational requirement brought about by the complex system model. By using CPU based sequential programming to implement the simulation, with the growth of converter levels, the higher execution time makes it impractical. With the development of GPGPU, the system-level converter structure makes it suitable for massively parallel implementation. Compared with single-core and multi-core CPU, the many core GPU structure brings unique computational capability for massive-thread parallel problem solution.

This thesis describes the nonlinear physics-based IGBT and power diode model and numerical solver implementation for power electronic circuit simulation. The device-level massive parallelism is implemented on these models and the solvers. In power electronic circuits simulation, the accomplishment of high level MMC gives opportunity for more massively parallel simulation application. The contributions of this thesis and recommendations for future work are presented in this chapter.

5.1 Contributions

- The device-level physics based IGBT and power diode model are linearized and discretized to be solved using linear solution method. The equivalent circuits of com-

plex nonlinear power electronic models are provided.

- IGBT and power diode and numerical solution have wide application in power electronic circuits. Massively paralleled implementation of the device-level models was proposed for the first time on the GPU. These modules can be used in future applications.

- The test cases of up to 3-phase 11-level MMC circuit simulation using physics-based models is hard to perform using SaberRD®. Using the methods in this thesis on GPU gives simulation results and achieves accelerated execution time.

- The dimension of Jacobian matrix grows with the level of MMC circuit. The block computation using partial LU decomposition proposed up in this thesis increases the system-level parallelism, which can help solve large power electronic circuits containing repeated structures.

- A comparison of LU decomposition and Gaussian elimination solution is developed for various applications. When using parallel implementation of the two methods, the execution procedures are closer than in sequential code, which makes the advantageous aspects of these two solution method suitable for specific cases with less limitation.

- The predictor-corrector variable time-step method increases the simulation efficiency by arranging computational resources according to tasks, which can benefit wide range of circuit simulation applications.

- Since large numbers of physics-based models in MMC will cause convergence problem, using the linear behavior-based IGBT and power diode models for up to 501-levels MMC with parallel simulation shows significant speed-up comparing with sequential simulation.

5.2 Future work

- The main limitation of application for higher MMC circuit simulation comes from the ill-conditioning of the Jacobian matrix. One approach is to increase the precision of calculation with the improvement of hardware.

- There are many ways to improve the Newton-Raphson method for nonlinear equations so that the convergence of the system can be increased.

- The physics-based IGBT and diode model can be improved using other discretization and linearization methods so that the Jacobian matrix of the IGBT can be better conditioned.
- The decomposition of large dimension Jacobian matrix in this thesis is based on a mathematical method. There are other ways such as using electric characteristics of the circuit and adding transmission lines to physically decompose the system.

Bibliography

[1] S. Kelkar, R. W. Wunderlich and L. Hitchcock, "Device level simulation for power converters", *in IEEE APEC Rec.,* 1989, pp. 335-343, 13-17 March 1989.

[2] L. Johnson, "Saber-MATLAB integrations: enabling virtual HW/SW co-verification", Synopsys, Inc., USA, June 2004

[3] "Analog and mixed signal simulation", Cadence Design Systems, Inc.

[4] "LTspice IV getting stated guide", Linear Technology, Inc., 2011

[5] C. Gu, "QLMOR: a projection-based nonlinear model order reduction approach using quadratic-linear representation of nonlinear systems", *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.,* vol. 30, no. 9, pp. 1307-1320, Sept. 2011.

[6] J. G. Linvil, "Lumped Models of Transistor and Diodes", *Proceedings of the IRE,* vol. 46, no. 9, pp. 1141-1152, June 1958.

[7] A. R. Hefner and D. M. Diebolt, "An experimentally verifed IGBT model implemented in the Saber circuit simulator", *IEEE Trans. Power Electron.,* vol. 9, no. 5, pp. 532-542, Sept. 1994.

[8] A. R. Hefner, "Modeling buffer layer IGBTs for circuit simulation", *IEEE Trans. Power Electron.,* vol. 10, no. 2, pp. 111-123, Mar. 1995.

[9] G. T. Oziemkiewicz, "Implementation and Development of the NIST IGBT model in a SPICE-based Commercial Circuit Simulator", *Univerisity of Florida MS Thesis,* Jan. 1996.

[10] K. Sheng B. W. Williams and S. J. Finney, "A review of IGBT model", *IEEE Trans. Power Electron.,* vol. 15, pp. 1250-1266, Nov. 2000.

[11] H. Shim, H. Kim, J. Song and D. Kim, "Analysis of high frequency characteristics of power inverters using accurate IGBT model based on datasheet and measurement",

Electrical Design of Advanced Packaging and System Symposium (EDAPS), 2015 IEEE pp. 81-84, Dec. 2015.

[12] J. T. Hsu and K. D. T. Ngo, "A behavioral model of the IGBT for circuit simulation", *Power Electron. Spec. Conf., 1995. PESC'95 Rec., 26th Annual IEEE, Atlanta, GA,* vol.2, pp. 865-871, Jun 1995

[13] Y. Y. Tzou and L. J. Hsu, "Practical SPICE macro models for the IGBT", *IECON Proc. (Ind. Electron. Conf.),* vol.2, pp. 762-766, Nov. 1993

[14] P. O. Lauritzen and C. L. Ma, "A simple diode model with reverse recovery", *IEEE Trans. Power Electron.,* vol.6, no. 2, pp. 188-191, Apr. 1991

[15] C. L. Ma and P. O. Lauritzen, "A simple diode model with forward and reverse recovery", *IEEE Trans. Power Electron.,* vol.8, no. 4, pp. 342-346, Oct. 1993

[16] "Saber® Accelerates Robust Design", Synopsys, Inc., USA, vol.5, June 2007

[17] NVIDIA Cormoration, "NIVIDA's Next Generation CUDA™ Compute Architecture : Kepler™ GK110", 2012.

[18] NVIDIA Cormoration, "NIVIDA's GeForce GTX 1080 Whitepaper", 2016.

[19] V. Jalili-Marandi and V. Dinavahi, "SIMD-Based large-scale transient stability simulation on the graphics processing unit", *IEEE Trans. on Power Syst., 2010,* vol. 25, no. 3, pp. 1589-1599, Aug. 2010.

[20] P. Bailey, J. Myre, S. D. C. Walsh, D. J. Lilha and M. O. Saar "Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors", *Int. Conf. on Parallel Processing, Vienna,* pp. 550-557, Sept. 2009.

[21] M. Smelyanskiy, D. Holmes, J. Chhugani and A. Larson "Mapping High-Fidelity Volume Rendering for Medical Imaging to CPU, GPU and Many-Core Architectures", *IEEE Trans. Vis. Comput. Graphics, 2009,* vol. 15, no. 6, pp. 1563-1570, Aug. 2010.

[22] J. D. Owens, M. Houston, D. Luebke and S. Green "GPU Computing", *Proc. IEEE,* vol.96, no. 5, pp. 879-899, May 2008.

[23] M. Rietmann, P. Messmer, T. Nessen-Meyer and D. Peter "Forward and adjoint simulations of seismic wave propagation on emerging large-scale GPU architectures",

International Conference for High Performance Computing, Storage and Analysis (SC'12), Salt Lake City, UT, pp. 1-11, Nov. 2012.

[24] V. Jalili-Marandi, Z. Zhou and V. Dinavahi, "Large-Scale Transient Stability Simulation of Electrical Power Systems on Parallel GPUs", *IEEE Trans. Parallel Distrib. Syst.,* vol.23, no. 7, pp. 1255-1266, July 2012.

[25] Z. Zhou and V. Dinavahi, "Parallel Massive-Thread Electromagnetic Transient Simulation on GPU," *IEEE Trans. Power Del.,* vol. 29, no. 3, pp. 1045-1053, June 2014.

[26] H. Karimipour and V. Dinavahi, "Extended Kalman Filter-Based Parallel Dynamic State Estimation," *IEEE Trans. on Smart Grid,* vol. 6, no. 3, pp. 1539-1549, May 2015.

[27] H. Karimipour and V. Dinavahi, "Parallel relaxation-based joint dynamic state estimation of large-scale power systems," *IET Generation, Transmission & Distribution,* vol. 10, no. 2, pp. 452-459, Feb. 2016.

[28] A. Gopal, D. Niebur and S. Venkatasubramanian, "DC Power Flow Based Contingency Analysis Using Graphics Processing Units," *Power Tech, 2007 IEEE Lausanne, Lausanne,* pp. 731-736, 2007.

[29] NVIDIA Corp., "CUDA C Programming Guide Version 5.5", July 2013.

[30] NVIDIA Corp., "Whitepaper NVIDIA's Next CUDA™ Compute Architecture Kepler™ GK110 V1.0",

[31] C. Gear, "Simultaneous Numerical Solution of Differential-Algebraic Equations," *IEEE Trans. on Circuit Theory,* vol. 18, no. 1, pp. 89-95, Jan 1971.

[32] S. Debnath, Q. Jiangchao, B. Bahrani, M. Saeedifard, and P. Barbosa, "Operation, control, and applications of the modular multilevel converter: a review," *IEEE Trans. Power Electron.,* vol.30, no. 1, pp. 37-53, Jan. 2015.

[33] H. Saad, S. Dennetière and J. Mahseredjian, "On modelling of MMC in EMT-type program" *IEEE 17th Workshop on Control and Modeling for Power Electronics (COMPEL),* pp. 1-7, Sept. 2016.

[34] D. C. Ludois and G. Venkataramanan, "Simplified terminal behavioral model for a modular multilevel converter," *IEEE Trans. on Power Electron.,* vol.29, no. 4, pp. 1622-1631, Apr. 2014.

[35] H. Peng, M. Hagiwara, and H. Akagi, "Modeling and analysis of switching-ripple voltage on the DC link between a diode rectifier and a modular multilevel cascade inverter (MMCI)," *IEEE Trans. on Power Electron.,* vol.28, no. 1, pp. 75-84, Jan. 2013.

[36] Z. Shen and V. Dinavahi "Real-Time Device-Level Transient Electrothermal Model for Modular Multilevel Converter on FPGA," *IEEE Trans. on Power Electron.,* vol.31, no. 9, pp. 6155-6168, Sept. 2016.

[37] H. Saad, J. Peralta, S. Dennetière; J. Mahseredjian, J. A. Martinez, A. Davoudi, M. Saeedifard, V. Sood, X. Wang, J. Cano and Ali MehriziSani, "Dynamic Averaged and Simplified Models for MMC-Based HVDC Transmission Systems", *IEEE Trans. on Power Del.,* vol.28, no. 3, pp. 1723-1730, Apr. 2013.

[38] M. Hagiwara; H. Akagi "Control and Experiment of Pulsewidth-Modulated Modular Multilevel Converters", *IEEE Trans. on Power Electron.,* vol.24, no. 7, pp. 1737-1746, July 2009.

[39] H. Akagi, S. Inoue and T. Yoshii "Control and Performance of a Transformerless Cascade PWM STATCOM With Star Configuration," *IEEE Trans. on Ind. Appl.,* vol. 43, no. 4, pp. 1041-1049, July-Aug. 2007.

[40] G. P. Adam, O. Anaya-Lara, G. M. Burt, D. Telford, B. W. Williams and J. R. Mcdonald , "Modular multilevel inverter: Pulse width modulation and capacitor balancing technique", *IET Power Electronics,* vol.3, no. 5, pp. 702-715, Aug. 2010.

[41] Y. Xue and Z. Xu, "On the Bipolar MMC-HVDC Topology Suitable for Bulk Power Overhead Line Transmission: Configuration, Control, and DC Fault Analysis," *IEEE Trans. on Power Del.,* vol. 29, no. 6, pp. 2420-2429, Dec. 2014.

[42] W. Li, L. Greigoire and J. Bélanger "Control and Performance of a Modular Multilevel Converter System," *CIGRÉ Canada, Conference on Power Systems,* Halifax, Sept. 2011.

[43] Ó. Jiménez, Ó. Luca, I. Urriza, L. A. Barragan, D. Navarro and V. Dinavahi "Implementation of an FPGA-Based Online Hardware-in-the-Loop Emulator Using High-Level Synthesis Tools for Resonant Power Converters Applied to Induction Heating Appliances," *IEEE Trans. on Ind. Electron.,* vol. 62, no. 4, pp. 2206-2214, April 2015.