Building Efficient and Reliable Emerging Technology Systems

by

Aporva Amarnath

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in the University of Michigan
2021

Doctoral Committee:

Associate Professor Ronald G. Dreslinski, Chair
Assistant Professor Baris Kasikci,
Assistant Professor Hun-Seok Kim,
Professor Wei Lu

Aporva Amarnath
aporvaa@umich.edu
ORCID iD: 0000-0002-5345-2022

To my family,
for their never-ending love and support.

ACKNOWLEDGMENTS

The support and contribution of several people have paved the path towards the completion of my thesis. I also recognize the grants and funding awarded by the Defense Advanced Research Projects Agency (DARPA) and the Electrical Engineering and Computer Science (EECS) Department at the University of Michigan towards my PhD work.

I am sincerely grateful to my advisor, Ronald Dreslinski, for giving me a chance to embark on the PhD journey. He is the sole reason I developed a passion for research and the one who taught me the art of it. He has always been a very supporting advisor who allowed me to shape my thesis to involve broad topics. I want to acknowledge Trevor Mudge for being the person who taught me how to do good research and the ability to tackle challenges with a top-down perspective. I only hope to be as enthusiastic about research as him in my professional career. I cannot emphasis the roles my favorite teachers, K. R. Anupama, Biju Raveendran and Mark Brehob, have played in shaping my zeal for computer architecture.

Through my internships, I have had the opportunity to work with inspiring mentors, John Kalamatianos from AMD and Augusto Vega, Alper Buyuktosunoglu, John-David Wellman, Hubertus Franke and Pradip Bose from IBM. Working with them has immensely expanded my horizon of knowledge. I would like to express my gratitude for their guidance over the years I have known them.

I have had the pleasure of working with amazing people who have become my friends, Tutu Ajayi, Jielun Tan, Siying Feng, Javad Bagherzadeh, Hiwot Kassa, Heewoo Kim, Deepika Natarajan, Nishil Talati, Austin Rovinski, Yichen Yang, Chris Torng and collaborators from many other universities. Working with them only made the intense work hours

ever more enjoyable. I want to thank my best friends, Aparajitha Chappa, Chithra Chandran, Gnanapriya Gnanaprakasam, Shreeranjani Srirangamsridharan, Balaramraju Buddharaju, Haritha Valluri and Mahathi Padegal for being extremely patient with me as I lived through a roller coaster of a PhD life. Talking to them would lighten up my day and I always had new rigor to tackle the next challenge.

I do not have enough words to express the gratitude I feel for my parents, Amarnath and Anitha. Their sacrifices, unconditional love and unwavering support makes me always strive for the best. My greatest source of inspiration, my best friend, my pillar of support and the best sister in the world, Anagha. I am always following her footsteps including earning a doctorate. I owe everything in my life to the three of them.

Finally, to the most important person in my life who made it all happen, my partner in crime and my best friend in life, Subhankar. He made this journey we embarked on ever so bearable. He has always been my strongest motivator and loudest cheer. I lovingly dedicate this thesis and life changing journey of mine to him.

TABLE OF CONTENTS

Dedication . . . . . **ii**

Acknowledgments . . . . . **iii**

List of Figures . . . . . **viii**

List of Tables . . . . . **xi**

Abstract . . . . . **xii**

Chapter

1 Introduction **1**

1.1 Contributions 2
1.2 Organization of the Dissertation 5

2 Carbon Nanotube Field-Effect Transistor **6**

2.1 Fabrication 6
2.2 Characterization 7
2.3 Need for Circuit-Level Enhancements 8
2.4 Variation 9
2.5 Need for Yield-Enhancing System-level Solution 10

3 Design Optimization Using Pass Transistor Logic **12**

3.1 Introduction 12
3.2 Motivation 13
3.2.1 CNFET Fabrication 14
3.2.2 CNFET Characterization 14
3.3 Related Work 16
3.4 RISC-V Processor Pipeline 18
3.5 Methodology 19
3.5.1 Operating Voltage 19
3.5.2 CNFET Design Parameters 19
3.5.3 Implementation 21
3.6 Evaluation 21
3.6.1 Adder Analysis 22
3.6.2 Multiplier 25

3.6.3 Registers . . . 26
3.6.4 Full Pipeline . . . 26
3.7 Conclusions . . . 28

4 A Design Framework for High-Variation Transistor Technology . . . 29

4.1 Motivation . . . 32
4.2 Variation Model . . . 33
4.2.1 CNT Distribution . . . 34
4.2.2 Variation Suite . . . 35
4.3 Architecture . . . 37
4.4 Design Flow for High-variation Technology . . . 40
4.4.1 Standard Cell Library . . . 40
4.4.2 Library Generation . . . 40
4.4.3 Design Methodology . . . 41
4.5 Evaluation . . . 42
4.5.1 Performance and EDP Analysis . . . 42
4.5.2 Frequency and Area Overhead . . . 44
4.6 Related Work . . . 44
4.7 Conclusion . . . 45

5 Scheduling Techniques for Real-Time constrained Heterogeneous SoCs . . . 47

5.1 Background and Motivation . . . 52
5.1.1 Autonomous Vehicle Applications . . . 52
5.1.2 Congestion in Environmental Conditions . . . 55
5.1.3 Application Deadline and Speed of the AV . . . 56
5.1.4 Quality-of-Mission (QoM) Metrics . . . 56
5.1.5 Domain-Specific Systems-on-Chips . . . 57
5.1.6 State-of-the-Art Real-Time *and* Heterogeneous Schedulers . . . 58
5.2 AVSched: Mission & Heterogeneity-Aware Scheduler . . . 60
5.2.1 Meta-Sched and Task-Sched Operations . . . 60
5.2.2 Summary of Cumulative AVSched Features . . . 68
5.3 Experimental Methodology . . . 69
5.3.1 Hardware Description . . . 69
5.3.2 Application Task Profile . . . 70
5.3.3 Energy and Power Model . . . 74
5.3.4 Trace generation . . . 75
5.3.5 Simulation Platform . . . 77
5.4 Evaluation . . . 77
5.4.1 Offline Profiling . . . 77
5.4.2 Single-Level Task Scheduler Evaluation . . . 78
5.4.3 AVSched Optimizations . . . 78
5.4.4 Scheduler Evaluation for Real-World Applications . . . 79
5.5 Related Work . . . 83
5.6 Conclusion . . . 85

6 Device-level Heterogeneous System . . . 87

6.1 Introduction . . . 87
6.2 Background and Motivation . . . 88
6.2.1 Task Requirements . . . 88
6.2.2 Emerging Compute Technologies . . . 88
6.2.3 FO4 Inverter characterization . . . 90
6.3 System Description . . . 91
6.3.1 SoC Design . . . 91
6.3.2 Scheduling Policy . . . 92
6.4 Methodology . . . 93
6.4.1 Spice Simulation . . . 93
6.4.2 Accelerator Characterization . . . 93
6.4.3 Heterogeneous System Setup . . . 94
6.4.4 Task Traces . . . 94
6.4.5 Metrics of Evaluation . . . 95
6.5 Evaluation . . . 95
6.5.1 Module Characterization . . . 95
6.5.2 Accelerator Characterization . . . 100
6.5.3 System Evaluation . . . 100
6.6 Conclusion . . . 101
7 Conclusion . . . 103
Bibliography . . . 104

LIST OF FIGURES

1.1 To overcome manufacturing issues in emerging technologies, solutions need to be developed across the stack. This work presents solutions that target varying levels of the computing stack 3

2.1 Comparison of FO4 inverter in CCNT and Si-CMOS 8
2.2 Compute size achieved for 99.73% yield if no reliability, core-level, pipeline-stage level, or module-level solution is employed for varying fault rates. Dashed line denotes the current process's failure rate of a CNFET (10 ppm). 10

3.1 Comparison of FO4 inverter in CCNT and Si-CMOS 16
3.2 Restoring logic for cascaded full adders 17
3.3 Varying pitch and width of the CNFET 20
3.4 Pass transistor-based full adder 22
3.5 Improvement of PTL-CNT and CCNT over silicon for a full adder 23
3.6 Improvement of PTL-CNT and CCNT over silicon for (a) ripple-carry adder, (b) Kogge-Stone adder and (c) V-scale ALU 24
3.7 Improvement of PTL-CNT and CCNT over silicon for the multiplier 26
3.8 Improvement of CCNT over silicon for the D-Flip Flop 27
3.9 Improvement of CCNT-PTL-CNT Hybrid and CCNT over silicon for the V-scale pipeline 28

4.1 Path to sustenance for a disruptive technology [1] 30
4.2 Compute size achieved for 99.73% yield if no reliability, core-level, pipeline-stage level, or module-level solution is employed for varying fault rates. Dashed line denotes the current process's failure rate of a CNFET (10 ppm). 32
4.3 Effect of CNT distribution (varying pitch sigma) on FO4 inverter 3σ delay and yield failures for W = 100 nm and s = 20 nm 35
4.4 3σ-delay derate of statistical model in comparison to actual derate for FO4 inverter chains, of drive strength 1, of varying lengths from 1 to 20. 36
4.5 Schematic of 3DTUBE, where identical components are stacked vertically, and crossbars are inserted between stages. In this four-failure scenario, a regular 3D CMP would only have one working core. In contrast, 3DTUBE dynamically reconfigures to connect healthy units creating 2 complete cores (red and green stripes). Stages in orange are idle units that could not be used to form functional pipelines. 38

4.6 a) 3DTUBE pipeline-level structure b) Parallel module-level architecture c) Serial module-level architecture 39
4.7 Performance of baseline, pipeline-stage level, and module level solutions of 3DTUBE for varying transistor failure rate. Dashed line denotes the current process's failure rate of a CNFET (10 ppm). Si-based design is evaluated at a failure rate of 1 ppb. 43

5.1 *Left-axis:* Mission speedup of Quality-of-Mission aware scheduler ("QoM-aware") over the Quality-of-Mission agnostic scheduler ("QoM-agnostic") and *Right-axis:* PE utilization of "QoM-agnostic" and "QoM-aware", when evaluated under three congestion scenarios: rural, semi-urban and urban. 49
5.2 AVSched's operation workflow, including the off-line profiling and run-time scheduling stages. 51
5.3 Use of AVSched to optimize SoC design for AVs while considering QoM metrics, PE utilization and energy consumption. Use of an efficient scheduler in this optimization loop helps reduce the compute requirements of an SoC while improving the mission performance 53
5.4 Effect of SoC heterogeneity, calculated as the coefficient of variation in PE peak performance, on mission speedup and PE utilization of a Quality-of-Mission-aware scheduler ("QoM-aware") over a Quality-of-Mission agnostic ("QoM-agnostic"). We increase heterogeneity in the system by considering varying PE configurations in the SoC. 57
5.5 AVSched operations showing `Meta-Sched` (mission & DAG processor) and `Task-Sched` (task scheduler & hardware manager), and their synchronization using the ready and completed queues, and prune list. 62
5.6 *Left:* A 7-task DAG containing three paths: P_0, P_1 and P_2. P_0 contains tasks 0, 2, 4 and 6; P_1 contains tasks 1, 4 and 6; and P_2 contains tasks 1, 3 and 5. *Right:* Timing profile for each task on three types of PEs: CPU, GPU and an accelerator. Using the timing profile, we determine that P_0 is the critical path, P_1 intersects the critical path, and P_2 is independent of it. 64
5.7 Incremental improvement in mission time and PE utilization on Sys_A of (a) 1-level task scheduler `TS` over `TS-greedy` (b) 2-level task schedulers with ranking (`MSx`) over `TS`, (c) hetero-ranking enabled `MSx` over `MSx`, (d) hier/hetero-ranking enabled `MSx` over hetero-ranked `MSx`, (e) update, hier/hetero-ranking enabled `MSx` over hier/hetero-ranked `MSx`. Note that MS1 is MS_{static} and MS2 is $MS_{dynamic}$ 74
5.8 Comparison of mission performance and PE utilization of AVSched against prior-work schedulers for (a) ADSuite, (b) 3D Mapping, and (c) Package Delivery. *Left:* Sys_B. *Right:* Sys_C. 80
5.9 *Top.* Slack, energy savings and mission overhead for AVSched with DVFS enabled across the three applications. *Bottom.* Per-PE utilization for each application. Results are shown for the rural congestion. The execution run corresponding to the f_{slack} value with best energy savings is reported here. Note that the utilization values for two accelerators (DET and TRA) are zero because the latter two applications do not have any tasks that use these PEs. . . 82

5.10 Normalized product of energy, mission time for varying SoC configurations for ADSuite, 3D Mapping and Package Delivery using AVSched in an urban scenario. We evaluate AVSched on SoC configurations that have varying CPU core count, GPU count, detection accel. count (for ADSuite – other PE types are at one count) and localization accel. count (for 3D Mapping and Package Delivery – other PE types are at zero count) to achieve the SoC configuration with the best energy and mission time (denoted in green). 84

6.1 Comparison of FO4 inverter in Si, CNFET and TFET. We evaluate (a) delay across all operable voltages, (b) the trade-off between power and delay, (c) trade-off between energy and delay and (d) trade-off between energy-delay product and delay. 90
6.2 System Composition. 92
6.3 Comparison of 32-bit integer adder in Si, CNFET and TFET. We evaluate (a) delay across all operable voltages, (b) the trade-off between power and delay, (c) trade-off between energy and delay and (d) trade-off between energy-delay product and delay. 96
6.4 Comparison of 32-bit integer multiplier in Si, CNFET and TFET. We evaluate (a) delay across all operable voltages, (b) the trade-off between power and delay, (c) trade-off between energy and delay and (d) trade-off between EDP and delay. 97
6.5 Comparison of single precision floating point adder in Si, CNFET and TFET. We evaluate (a) the trade-off between power and delay, (b) trade-off between energy and delay and (c) trade-off between EDP and delay. 98
6.6 Comparison of single precision floating point multiplier in Si, CNFET and TFET. We evaluate (a) delay across all operable voltages, (b) the trade-off between power and delay, (c) trade-off between energy and delay and (d) trade-off between EDP and delay. 99
6.7 Comparison of metrics for (a) single precision `fft` accelerator, (b) 32-bit integer `gemm` and (c) 32-bit integer `stencil` accelerator. CNFET-based designs are compared against Si-based design operating at the best EDP. TFET-based designs are compared against Si-based design operating at the best energy. . . 100
6.8 Comparison of response time for `Type-D` tasks, energy for `Type-E` tasks and EDP for `Type-ED` tasks of hetero-system over baseline Si-system. 101
6.9 Comparison of (a) total execution time and (b) energy consumption of the hetero-system over Si-system for varying task arrival rates. Note that the hetero-system employs fine grained power management to reduce idle leakage energy. 102

LIST OF TABLES

3.1 Ripple-carry adder design results 25
3.2 Kogge-Stone adder design results 25
3.3 V-scale ALU results 25
3.4 Array multiplier results 26
3.5 V-scale pipeline results 28

4.1 Synthesis results of an OpenSPARC T1 core optimized for delay 41
4.2 EDP reduction of 3DTUBE for an 8-core design at 10 ppm failure rate over the Si-based design evaluated at 1 ppb failure rate 42

5.1 Real-Time and Heterogeneous Schedulers. 58
5.2 Description of parameters used in AVSched. 60
5.3 System Description of SoC for Workload Evaluation. 70
5.4 Timing and power profile of evaluated kernels on each PE-type. 76

ABSTRACT

The semiconductor industry has been reaping the benefits of Moore's law powered by Dennard's voltage scaling for the past fifty years. However, with the end of Dennard scaling, silicon chip manufacturers are facing a widespread plateau in performance improvements. While the architecture community has focused its effort on exploring parallelism, such as with multi-core, many-core and accelerator-based systems, chip manufacturers have been forced to explore beyond-Moore technologies to improve performance while maintaining power density. Examples of such technologies include monolithic 3D integration, carbon nanotube transistors, tunneling-based transistors, spintronics and quantum computing. However, the infancy of the manufacturing process of these new technologies impedes their usage in commercial products. The goal of this dissertation is to combine both architectural and device-level efforts to provide solutions across the computing stack that can overcome the reliability concerns of emerging technologies. This allows for beyond-Moore systems to compete with highly optimized silicon-based processors, thus, enabling faster commercialization of such systems. This dissertation proposes the following key steps: (i) Multifaceted understanding and modeling of variation and yield issues that occur in emerging technologies, such as carbon nanotube transistors (CNFETs). (ii) Design of systems using suitable logic families such as pass transistor logic that provide high performance. (iii) Design of a multi-granular fault-tolerant reconfigurable architecture that enhances yield and performance. (iv) Design of a multi-technology, multi-accelerator heterogeneous system (v) Development of real-time constrained efficient workload scheduling mechanism for heterogeneous systems. This dissertation first presents the use of pass

transistor logic family as an alternate to the CMOS logic family for CNFETs to improve performance. It explores various architectural design choices for CNFETs using pass transistor logic (PTL) to create an energy-efficient RISC-V processor. Our results show that while a CNFET RISC-V processor using CMOS logic achieves a 2.9× energy-delay product (EDP) improvement over a silicon design, using PTL along the critical path components of the processor can boost EDP improvement by 5× as well as reduce area by 17% over 16 nm silicon CMOS. This document further builds on providing fault-tolerant and yield enhancing solutions for emerging 3D integration compatible technologies in the context of CNFETs. The proposed framework can efficiently support high-variation technologies by providing protection against manufacturing defects at multiple granularities: module and pipeline-stage levels. Based on the variation observed in a synthesized design, a reliable CNFET-based 3D multi-granular reconfigurable architecture, 3DTUBE, is presented to overcome the manufacturing difficulties. For 0.4-0.7 V, 3DTUBE provides up to 6.0× higher throughput and 3.1× lower EDP compared to a silicon-based multi-core design evaluated at 1 part per billion transistor failure rate, which is 10,000× lower in comparison to CNFET's failure rate. This dissertation then ventures into building multi-accelerator heterogeneous systems and real-time schedulers that cater to the requirements of the applications while taking advantage of the underlying heterogeneous system. We introduce optimizations like task pruning, hierarchical hetero-ranking and rank update built upon two scheduler policies (MS_{static} and $MS_{dynamic}$), that result in a performance improvement of 3.5× (average) for real-world autonomous vehicle applications, when compared against state-of-the-art schedulers. Adopting insights from the above work, this thesis presents a multi-accelerator, multi-technology heterogeneous system powered by a multi-constrained scheduler that optimizes for varying task requirements to achieve up to 6.1× better energy over a baseline silicon-based system.

CHAPTER 1

Introduction

Gordon Moore made the observation that as the transistor technology scales there will be roughly double the number of transistors every two years improving performance. Dennard scaling stated that the power density of a chip remains same with the reduction of transistor size by half. However, around the early 2000's we saw the end of Dennard scaling which had guaranteed constant power density as the technology scales. This coupled with the thermal and power limits led to single threaded performance and frequency stagnation. Although the industry recuperated by relying on both architectural and device level innovations [2,3,4,5,6] to sustain Moore's Law like multicore processors, currently the power constraints limit us from utilizing all the cores on the processor. Moreover, with physical scaling of silicon (Si) predicted to end with the 5 nm node [7], extensive research is being conducted to incorporate specialized hardware and the use of beyond-Moore technologies to sustain Moore's law. We have seen the emergence of various new technologies that continue performance scaling while maintaining power density and can either supplement or replace silicon-based transistors. Some prominent disruptive technologies are carbon-nanotube transistor (CNFET), germanium-nanowire transistor, silicon carbide transistor and tunnel transistor (TFET). However, there is no alternative technology at the moment that has the capability to match the yield and performance of Si for existing designs. Furthermore, due to the infancy of their manufacturing process, high defect densities, and variation issues, chip designers are not encouraged to consider these emerging technologies as a stand-alone

replacement for Si-based transistors. Extensive research on both the manufacturing as well as architecture fronts are required to move innovation forward to create large scale chips using these new technologies. However, aggressive manufacturing research will not be done unless a product is marketed, and products cannot be developed profitably because of the high variation in the manufacturing process, leading to a viscous cycle. The vision of this dissertation is to develop solutions for high variation emerging technologies that can overcome the reliability concerns of a new technology and compete with the highly optimized state-of-the-art silicon-based processors, enabling faster commercialization of these technologies to develop a competitive device-level heterogeneous system.

To accomplish this goal, we must overcome several challenges. The largest of these barriers is related to high fault rates and yield issues observed in new technologies. For example, carbon nanotube transistors, a prominent emerging technology, is affected by manufacturing variability [8]. The latest process used to create carbon nanotube-based designs demonstrates a high failure rate of 10 ppm (parts per million) [9]. As a result, to achieve a yield of 99.73% (3-sigma process), it can only realize a maximum design size of less than 300 transistors, which is negligible in comparison to the 800 million transistors in Intel's Core i7 processor. As shown in Figure 1.1, our proposed research calls for solutions across the computing stack to efficiently deploy emerging technologies; build detailed variation models, apply circuit level logic family optimizations, develop a fault-tolerant multi-granular reconfigurable architecture and build an efficient function-level and device-level heterogeneous system with effective real-time workload scheduling to achieve high performance at a low energy cost.

1.1 Contributions

This dissertation proposes the use of pass transistor logic family as an alternate to complementary metal-oxide semiconductor (CMOS) logic family for CNFETs to improve

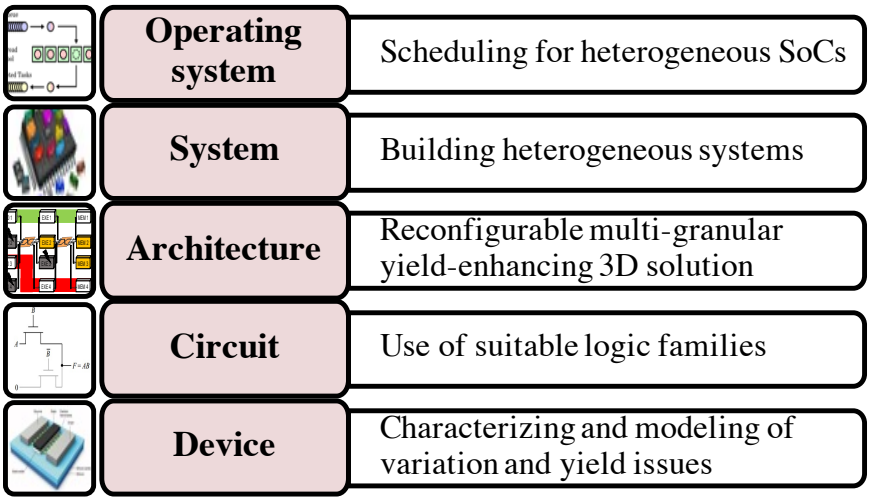


Figure 1.1: To overcome manufacturing issues in emerging technologies, solutions need to be developed across the stack. This work presents solutions that target varying levels of the computing stack

performance [10]. Although logic gates using CNFETs have been demonstrated to provide up to an order of magnitude better energy-delay product (EDP) over silicon-based counterparts, system-level design using CNFETs show significantly smaller EDP improvement because of manufacturing issues, critical path of the design, output load capacitance and corresponding drive strengths of gates. This work addresses this challenge by exploring various architectural design choices using CNFET-based pass transistor logic (PTL) and create an energy-efficient RISC-V processor. While silicon-based design traditionally prefers complementary logic over PTL, CNFETs are ideal candidates for PTL due to their low threshold voltage, low power dissipation, and equal strength p-type and n-type transistors.

Furthermore, a design flow framework that can be used for large-scale chip production while mitigating yield and variation failures to bring up CNFET-based technology, using a reliable reconfigurable architecture is proposed. Although CNFET is one of the most promising competing technologies available, offering exceptional electrostatic properties, due to the infancy of their manufacturing process, high defect densities, and variation issues, chip designers are not encouraged to consider these emerging technologies as a stand-alone replacement for Si-based transistors. Hence, to commercialize these new

technologies, new architectural and circuit modifications that can work around high-fault rates are required, improving performance comparable to Silicon, while the manufacturing process is perfected. This work leverages the fact that CNFETs are perfect candidates for 3D integration due to their low-temperature manufacturing process and low power consumption. The proposed framework can efficiently support high-variation technologies by providing protection against manufacturing defects at multiple granularities: module and pipeline-stage levels [11]. To incorporate different CNFET manufacturing processes, this work also builds a flexible variation model and a CMOS-based CNT design library that can be used to synthesize physical CNFET-based processor designs over a range of 0.4 to 0.7 V.

Heterogeneous SoCs for autonomous vehicles (AVs), while necessary to meet stringent performance and safety constraints, pose challenges for traditional task scheduling approaches. In this work, we present AVSched, a multi-level scheduler that exploits the highly heterogeneous nature of the underlying SoC in conjunction with the characteristics of an AV application [12]. AVSched's goal is to improve a global objective function, exemplified by a defined *Quality-of-Mission* (QoM) metric, providing a more *holistic* scheduling approach that investigates the full hardware-software AV stack to improve the overall mission's quality rather than focusing solely on the real-time requirements of individual kernels or applications. Our evaluation shows that AVSched improves overall mission performance by an average of **5.4**×, **3.2**×, **2.9**× and **2.9**× when compared to CPATH, RHEFT, 2lvl-EDF and ADS (current, state-of-the-art real-time heterogeneous schedulers).

While heterogeneous SoCs are developed to cater to growing requirements of highly heterogeneous applications, prior art has explored heterogeneity either at the function-level or the device-level. This work explores combining the two to cater to performance and energy requirements of common kernels in both server and embedded system applications. In this work, we presented a function-level and device-level heterogeneous SoC, S_{hetero}, built to accelerate kernels using three different device technologies of silicon FETs, carbon nanotube FETs and tunnel FETs. The goal of the work is to cater to performance, energy

and energy-delay requirements of tasks using the different accelerators built using each device technology based on their operational strengths. We show that S_{hetero} in combination with a scheduler-driven sleep-based power optimization allows for a 1.7-6.2$\times$ improvement in system energy for varying task traces and arrival rates over a homogeneous Si-based system with DVFS enabled.

1.2 Organization of the Dissertation

The rest of the document is organized as follows. *Chapter 2* presents the key characteristics and challenges of carbon nanotube-based transistors and motivates this dissertation. *Chapter 3* proposes circuit level design optimizations to reap the potential benefits of CNFETs [10]. *Chapter 4* builds a variation model and design library for carbon nanotubes that are used to present 3DTUBE, a multi-granular yield-enhancing reconfigurable 3D architecture [11]. *Chapter 5* presents a scheduling mechanism for heterogeneous system-on-chip (SoC) architectures in the presence of real-time constraints. *Chapter 6* presents the design of a device-level heterogeneous system and evaluates it for various input requirements and optimizations. *Chapter 7* summarizes and concludes the dissertation.

All the work presented in this document has been done in collaboration with Subhankar Pal, Siying Feng, Tutu Ajayi, Austin Rovinski, Hiwot Kassa, Javad Bagherzadeh and Jielun Tan from University of Michigan and Augusto Vega, Alper Buyuktosunoglu, John-David Wellman, Hubertus Franke and Pradip Bose from IBM Corp.

CHAPTER 2

Carbon Nanotube Field-Effect Transistor

A carbon nanotube is a cylindrical rolled-up structure of a carbon hexagonal layer. A carbon nanotube transistor is a nanowire transistor with gate-all-around structure. Due to their similarities to Silicon-based Field-Effect Transistor (Si-FET), CNFETs exhibit a linear region followed by a saturation region in the drain current, I_{DS}, as a function of increasing gate-source voltage, V_{GS} [13]. CNFETs are one of the most promising competing technologies available, offering high current-carrying capacity [14], high carrier velocity [15], and exceptional electrostatics due to their ultra-thin body [16]. High mobility makes CNFETs have low latency with the same dynamic power dissipation. The gate-all-around structure enhances the gate controllability to channel potential, which results in steep on-off switching and low leakage power dissipation. Moreover, the low process temperature makes it possible for CNFETs to be used in monolithic 3D integration [17]. The high thermal conductivity helps CNFET mitigate the power burden of 3D integration.

CNFETs have been shown to be excellent candidates for low voltage and near threshold operations making them perfect candidates to be used in the design of sensors, IoT devices and energy-constrained devices.

2.1 Fabrication

Historically, CNFET designs have been plagued by manufacturing issues, particularly when creating a standard cell-based design. However, recent advances in fabrication techniques

have made high-yield, reliable CNFET devices possible for both p-type and n-type transistors, enabling the use of traditional Computer-Aided Design (CAD) flows.

Although CNFETs have faced several difficulties in efficient fabrication, recent techniques have improved the feasibility of CNFET manufacturing. Shulaker *et al.* have demonstrated highly aligned CNTs with a density (ρ_{cnt}) of about 100 CNTs/μm through chemical vapor deposition. Their method involves growing CNTs on a quartz substrate and repeatedly transferring them onto a wafer [18]. Hongsik *et al.* propose a technique where they fabricate and purify CNTs separately and suspend them on the substrate. Following this, they attract the CNTs into adhesive-filled trenches for alignment, resulting in a yield density of 20 CNTs/μm [19]. Recently, Brady *et al.* have achieved a $\rho_{cnt} \approx 50$ CNTs/μm using the floating evaporative self-assembly (FESA) method [20]. Franklin *et al.* [21] characterize multiple FETs fabricated with varying width from 3 μm to 15 nm on one CNT. Data extracted from these FETs are used to make more realistic CNFET models [22].

2.2 Characterization

While integrated circuits are predominantly composed of Si-CMOS, CNFETs offer a large number of advantages. In this section, we seek to quantify these benefits to understand how CNFETs can be leveraged over Si-CMOS logic.

To investigate the characteristics of CNFETs, we compare CNFET with CMOS (CCNT) to Si with CMOS (Si-CMOS) by using SPICE models of a minimum-sized 16 nm Si-CMOS inverter and an equivalent width 16 nm CCNT inverter. In Figure 2.1, we demonstrate the performance of the CNFET inverter using fan-out-of-four (FO4) analysis. Our characterization in Figures 2.1(a)-(d) shows that CNFETs outperform silicon both in terms of energy and EDP across the voltage range. However, CNFETs under-perform in comparison to Si-CMOS in FO4 delay at higher supply voltages due to the high contact resistance in CNFETs. This changes at lower voltages (approaching 0.4 V), where CNFETs edge out

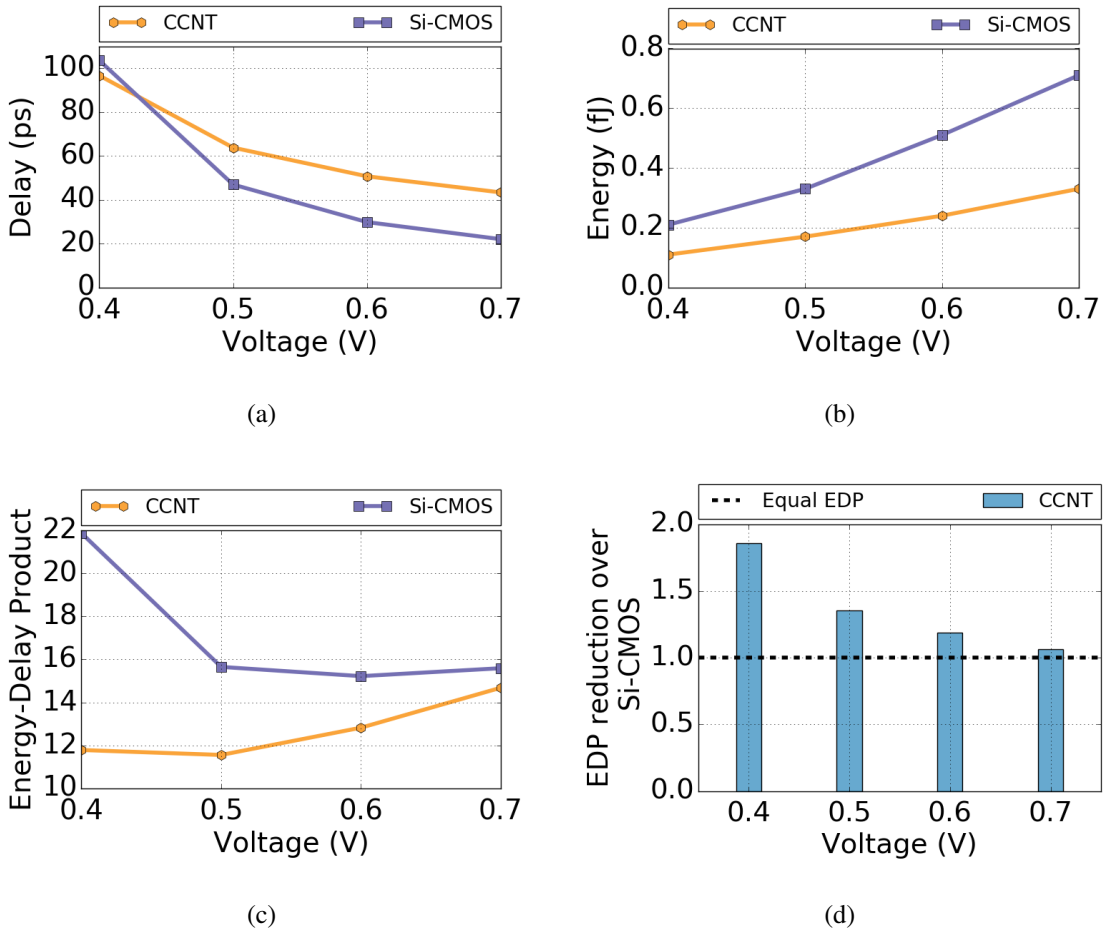


Figure 2.1: Comparison of FO4 inverter in CCNT and Si-CMOS

Si-FETs, because of CNFETs' higher current properties at lower voltages. Figure 2.1(d), in particular, shows that as the supply voltage decreases, the EDP advantage of CNFET over Si-FET increases.

2.3 Need for Circuit-Level Enhancements

While previous works have theorized up to an order of magnitude in EDP improvement for a CCNT-based inverter over Si-CMOS at low voltages [21, 23], they used theoretical models that did not include factors such as high contact resistance and variable CNT pitch, which are present in CNFETs that can be fabricated today. These properties limit the gains of CNFETs

to less than the theoretical numbers. Overall, we observed a 1.8× improvement in EDP at 0.4 V using models based on experimental data. Hence, this calls for more efficient design techniques and a better-suited logic family to reclaim the order of magnitude improvements that CNFETs can deliver. One of the key properties of CNFETs is their low threshold voltage and low power dissipation, which lends very well to the use of a more efficient logic family like pass transistor logic (PTL) [24]. CNFET-based systems can greatly improve EDP through the use of multiple logic families, and in particular with the use of PTL [24].

2.4 Variation

Recent fabrication techniques have helped make large scale CNFET manufacturing processes possible. However, CNFET manufacturing yields imperfections, due to the presence of metallic carbon nanotubes (m-CNTs), imperfect m-CNT removal processes, chirality drift, CNT doping variations in the source/drain extension regions, and density fluctuations due to non-uniform inter-CNT spacing. Chemical synthesis of CNTs do not provide precise control over the locations of individual CNTs and consequently, significant variations can exist in the spacing between CNTs. This non-uniformity, which is expressed as CNT count density variation, leads directly to spatial non-uniformity in the electronic properties of CNFETs, and thus increased delay, signal level attenuation and failure. Moreover, a single defective CNFET can cause faults on the gate level, such as higher leakage, less balanced rise/fall delays or too much driver strength for either pull-up/pull-down path. However, one of the most critical manufacturing issues and a major contributor to delay variation is the presence of density variations in CNT growth. The lack of precise growth or placement of CNT on a wafer along with the removal of m-CNTs lead to high variation in the CNT density. Leveraging the availability of theoretical CNFET models, prior works have conducted extensive studies on the impact of CNFET variations on yield and performance [8, 25]. In particular, they focused on the variation associated with CNT count in a transistor [26, 27, 28],

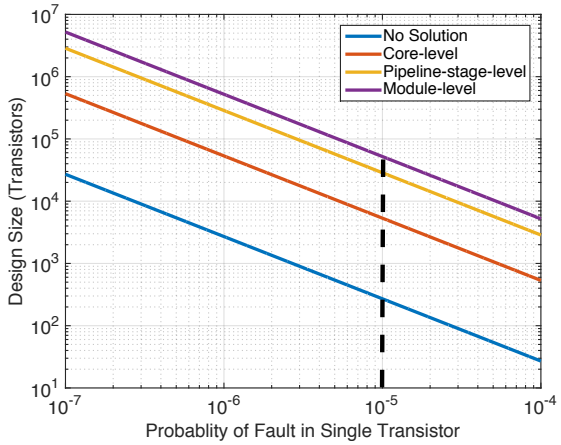


Figure 2.2: Compute size achieved for 99.73% yield if no reliability, core-level, pipeline-stage level, or module-level solution is employed for varying fault rates. Dashed line denotes the current process's failure rate of a CNFET (10 ppm).

which was identified as the major contributor to the delay variation in CNFETs [29]. Prior work has also been shown to improve the yield of the process at the layout or circuit level [9, 23, 27]. Stanford's Robust System Group also released a variation-aware Design Kit [30]. However, these lower-level solutions target a particular cause of variation and are not generic in nature and cannot be easily scaled to other emerging technologies. This calls for a system level yield enhancing solution that can be used to model and countermeasure any category of CNFET variation or reliability problem.

2.5 Need for Yield-Enhancing System-level Solution

For a technology to be deployed in a large-scale production, a high design yield of 3-sigma or above is required for the product to be profitable. However, due to the infancy of manufacturing processes of emerging technologies, we cannot achieve this yield without compromising on compute area or size of design.

CNFETs are affected by manufacturing variability [8], and several significant challenges must be conquered before the benefits of CNFETs can be reaped. The latest process used to create carbon nanotube-based designs demonstrates a high failure rate of 10 ppm (parts per

million) [8]. As can be seen in the Figure 2.2, for a process technology with a failure rate of 10 ppm to achieve a yield of 99.73% (3-sigma process), it can only realize a maximum design size of less than 300 transistors, which is negligible in comparison to the 800 million transistors in the Intel Core i7 processor.

Deriving motivations from the row/column redundancy utilized in SRAM arrays [31], we observe that by adding redundancy to the design, we can overcome the challenge of low yield and realize larger designs. Just adding the redundancy of a second core improves the design size by 20x as shown in Figure 2.2. Further, adding one additional spare at each pipeline-stage level and module level improves the design size by 190x and 255x, respectively, over the standalone, no reliability solution process. However, naively adding redundant cores, pipeline stages or modules adds a very expensive area overhead. Hence, gaining insight from these results, we later demonstrate, an efficient multi-core 3D architecture that can be reconfigured at multiple granularities to provide an inherent redundancy to improve the yield and performance of CNFET designs at a very small area overhead.

CHAPTER 3

Design Optimization Using Pass Transistor Logic

3.1 Introduction

With the end of Dennard Scaling and the pending demise of Moore's Law, silicon chip manufacturers are facing a widespread plateau in performance improvements. Clock frequencies and power have already stopped scaling due to the power wall [32], and many industry experts predict physical scaling to end with the 5 nm node in 2021 [7].

Extensive research is being undertaken towards the discovery of new alternative technologies to continue performance scaling while maintaining power density, including spintronics, quantum computing, and carbon nanotubes. CNFETs are one of the most promising competing technologies available, offering high current-carrying capacity [14], high carrier velocity [15], and exceptional electrostatics due to their ultra-thin body [16]. In addition, CNFETs have made great strides in manufacturability in terms of both device scaling and yield, and they require relatively few changes to the silicon manufacturing process [20].

Prior work has investigated the impact of CNFETs on small-scale designs, such as individual transistor properties or complementary gates [33, 34]. Bobba *et al.* have explored the impact of replacing Si-FETs with CNFETs at the system level, designing an OpenRISC processor [23]. However, their processor's EDP improvement is much lower in comparison to their gate-level EDP reduction over silicon. This is primarily due to the critical paths within the design, output load capacitance and corresponding drive strengths of gates while creating larger designs. The EDP improvement at system-level will further be diminished

due to variation caused by the fabrication process. Hence, this calls for more efficient design techniques and a better-suited logic family to reclaim the order of magnitude improvements that CNFETs can deliver. One of the key properties of CNFETs is their low threshold voltage and low power dissipation, which lends very well to the use of a more efficient logic family like pass transistor logic (PTL) [24]. CNFET-based systems can greatly improve EDP through the use of multiple logic families, and in particular with the use of PTL.

In this work, we take advantage of CNFETs' exceptional electrical properties to explore the architectural design considerations that need to be made when creating large-scale CNFET designs using PTL. We build a RISC-V pipeline using both complementary logic and PTL. Specifically, we compare several microprocessor components in 16 nm finFET-based CMOS silicon (Si-CMOS), 16 nm CMOS CNFET (CCNT), and 16 nm PTL CNFET (PTL-CNT). We then expand our analysis to a full RISC-V pipeline design and evaluate the system-level impacts.

We show that the CNFET RISC-V pipeline achieves a mere 2.9× improvement in energy-delay product over a silicon-based design at 0.4 V. We improve this by using PTL for the critical path components and CCNT for the rest of the design, gaining a 5× improvement in EDP and a 17% reduction in area over 16 nm silicon CMOS.

3.2 Motivation

Historically, CNFET designs have been plagued by manufacturing issues, particularly when creating a standard cell-based design. However, recent advances in fabrication techniques have made high-yield, reliable CNFET devices possible for both p-type and n-type transistors, enabling the use of traditional CAD design flows. CNFETs use carbon nanotubes as the channel medium between the source and the drain, instead of silicon. Hence, the behavior of a CNFET is similar to a Si-FET: we observe a linear region followed by a saturation region in the drain current, I_{DS}, as a function of increasing gate-source voltage, V_{GS} [13].

In this section, we briefly discuss recent fabrication breakthroughs, provide an initial characterization of the device, and demonstrate why PTL is a promising logic family for CNFET-based designs.

3.2.1 CNFET Fabrication

Although CNFETs have faced several difficulties in efficient fabrication, recent techniques have improved the feasibility of CNFET manufacturing. Shulaker *et al.* have demonstrated highly aligned CNT with a density (ρ_{cnt}) of about 100 CNT/μm through chemical vapor deposition. Their method involves growing CNT on a quartz substrate and repeatedly transferring them onto a wafer [18]. Hongsik *et al.* propose a technique where they fabricate and purify CNT separately and suspend them on the substrate. Following this, they attract the CNT into adhesive-filled trenches for alignment, resulting in a yield density of 20 CNT/μm [19]. Recently, Brady *et al.* have achieved a $\rho_{cnt} \approx 50$ CNT/μm using the floating evaporative self-assembly method [20]. Franklin *et al.* [21] characterize multiple FETs fabricated with varying width from 3 μm to 15 nm on one CNT. Data extracted from these FETs are used to make more realistic CNFET models [22].

3.2.2 CNFET Characterization

While integrated circuits are predominantly composed of Si-CMOS, CNFETs offer a large number of advantages. In this section, we seek to quantify these benefits to understand how CNFETs can be leveraged over Si-CMOS logic.

3.2.2.1 Complementary Logic

To investigate the characteristics of CNFETs, we compare CCNT to Si-CMOS by using SPICE models of a minimum-sized 16 nm Si-CMOS inverter and an equivalent width 16 nm CCNT inverter. In Figure 3.1, we demonstrate the performance of the CNFET inverter using fan-out-of-four (FO4) analysis. Our characterization in Figures 3.1(a)-(d) shows that

CNFETs outperform silicon both in terms of energy and EDP across the voltage range. However, CNFETs under-perform in comparison to Si-CMOS in FO4 delay at higher supply voltages due to the high contact resistance in CNFETs. This changes at lower voltages (approaching 0.4 V), where CNFETs edge out Si-FETs, because of CNFETs' higher current properties at lower voltages. Figure 3.1(d), in particular, shows that as the supply voltage decreases, the EDP advantage of CNFET over Si-FET increases.

While previous work has theorized up to an order of magnitude in EDP improvement for a CCNT-based inverter over Si-CMOS at low voltages [21, 23], they used theoretical models that did not include factors such as the contact resistance and variable CNT pitch, which are present in CNFETs that can be fabricated today. These properties limit the gains of CNFETs to less than the theoretical numbers. Overall, we observed a 1.8× improvement in EDP using models based on experimental data at 0.4 V.

3.2.2.2 Pass Transistor Logic (PTL)

Traditionally, Si-FET designs avoid using PTL because of the rapid threshold voltage drop across each additional PTL gate. Restoring logic is often used to balance this drop, however this negates the area, energy, and delay benefits of PTL. CNFETs possess three key properties that Si-FETs do not: CNFETs have a *very low threshold voltage*, while having a *low power dissipation* and *equal strength PFETs and NFETs*. With these key properties, CNFETs have been shown to enable PTL as a viable logic family [24].

However, to build larger designs using PTL, restoring logic is required. Figure 3.2 demonstrates the impact of using PTL with CNFETs. We show the number of stages after which a restoring buffer needs to be placed for cascaded full adders in both PTL-CNT and PTL-Si. For silicon, PTL requires frequent restoring logic (every 2-4 stages), which only worsens as the supply voltage decreases. PTL-CNT, however, requires much less frequent buffering due to its low threshold voltage and requires 6× fewer buffers than PTL-Si at 0.4 V. The buffering for PTL-CNT worsens as voltage increases, due to high contact resistance in

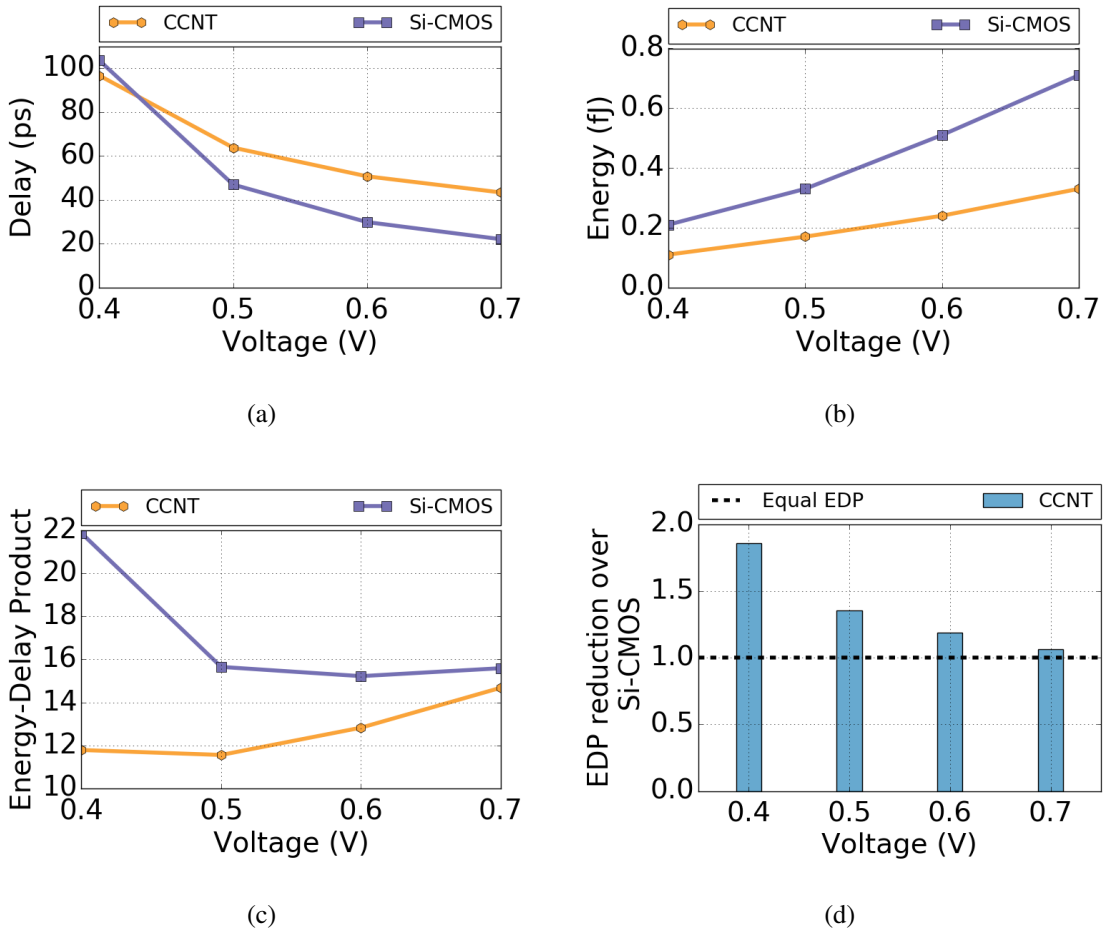


Figure 3.1: Comparison of FO4 inverter in CCNT and Si-CMOS

the CNFETs, although the total amount of required buffering remains superior.

From this initial characterization, we find that CNFETs outperform comparable Si-FETs in terms of EDP and are more amenable to PTL.

3.3 Related Work

Leveraging the availability of theoretical CNFET models, prior works have constructed the basic building blocks of a processor using CNFETs. Cho *et al.* [33] compare various CNFET-based standard cells against their counterparts made using Si-CMOS. Kumar *et al.* [34] propose a low-power full adder using CNFETs, showing an 80% power reduction

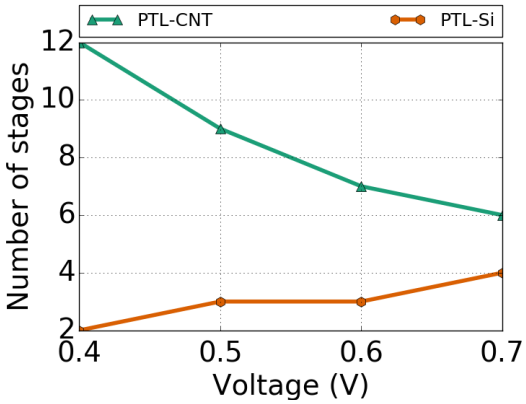


Figure 3.2: Restoring logic for cascaded full adders

in comparison to a Si-CMOS based one. Most of the work, however, has either been fragmented at the transistor-level or involved small building blocks.

In the work by Ding *et al.* [24], the authors explore building basic PTL gates using CNFETs. They also calculate the output voltage levels of a PTL-CNT single-bit adder and subtractor, and demonstrate a functional multiplexer and D-latch. However, their work neither studies scaling PTL to larger blocks, nor the challenges that accompany it.

Prior works have designed full systems based on CNFET technology. In the work by Shulaker *et al.* [35], the authors fabricate and demonstrate a functional, Turing-complete, subneg-based one-instruction-set computer at 1 μm. Further, Bobba *et al.* [23] show a 1.5$\times$ improvement in EDP of an OpenRISC processor, built using yield-enhancing standard cells, over Si-CMOS at 16 nm. However, these do not investigate the potential EDP improvement in system-level design that CNFETs provide in gate-level designs, nor do they explore the benefits of a suitable logic family, like PTL.

Our work is the first of its kind to construct an entire CNFET-based RISC-V processor with all its critical-path components such as the full adder, ALU, multiplier, and registers using PTL-CNT. We employ a pessimistic CNFET model to account for process variation, yet are able to demonstrate EDP improvements exceeding those that have been reported previously [23].

3.4 RISC-V Processor Pipeline

To address the challenges of system-level design and optimize CNFET-based systems, we build a single processor pipeline design using 3 different techniques: Si-CMOS, CCNT, and a hybrid (CCNT + PTL-CNT) configuration.

For our analysis, we use the V-scale core, which is a 32-bit, single-issue, in-order, 3-stage pipelined processor [36]. V-scale is an open-source design implemented in Verilog and is comparable to an ARM Cortex-M0 core. It is based on the open RISC-V instruction set architecture [37]. The critical modules of the core are implemented in each of the chosen configurations (Si-CMOS, CCNT and PTL-CNT) and then integrated into the full system.

The processor's ALU performs 14 different operations, including add/subtract, shift and comparison. We first implement the full adder circuit in the three different configurations. For comparison, we implement the 32-bit adder both as a ripple-carry and a Kogge-Stone design. A ripple-carry adder (RCA) consists of 32 full adders cascaded one after another and a Kogge-Stone adder (KSA) is a tree implementation of the carry-look ahead adder. While the KSA is faster and more energy-efficient than an RCA, it has a larger routing congestion and area [38]. Therefore, most present-day processors use sparse-tree adders that are a hybrid of both KSA and RCA. However, PTL implementations of these adders require custom addition of restoring logic between the stages, as discussed in Section 3.2.2.2, due to varying loads seen by each transistor, especially for sparse-tree adder designs closer to a KSA.

The multiplier is implemented as a 32-bit, two-stage array-based pipelined multiplier. It uses carry-save adders, which are a row of full and half adders cascaded one after another. As with the ripple-carry adder, the multiplier unit also requires restoring logic in the carry-save adders when implemented in PTL. These buffer insertions are periodic and are placed optimally to reduce the critical path delay and energy.

3.5 Methodology

This section details the design methodology used for our evaluation. We include descriptions of how our models were created and how we leveraged them to build standard cell libraries. Finally, we detail how we use those libraries to create custom blocks and the final V-scale pipeline.

3.5.1 Operating Voltage

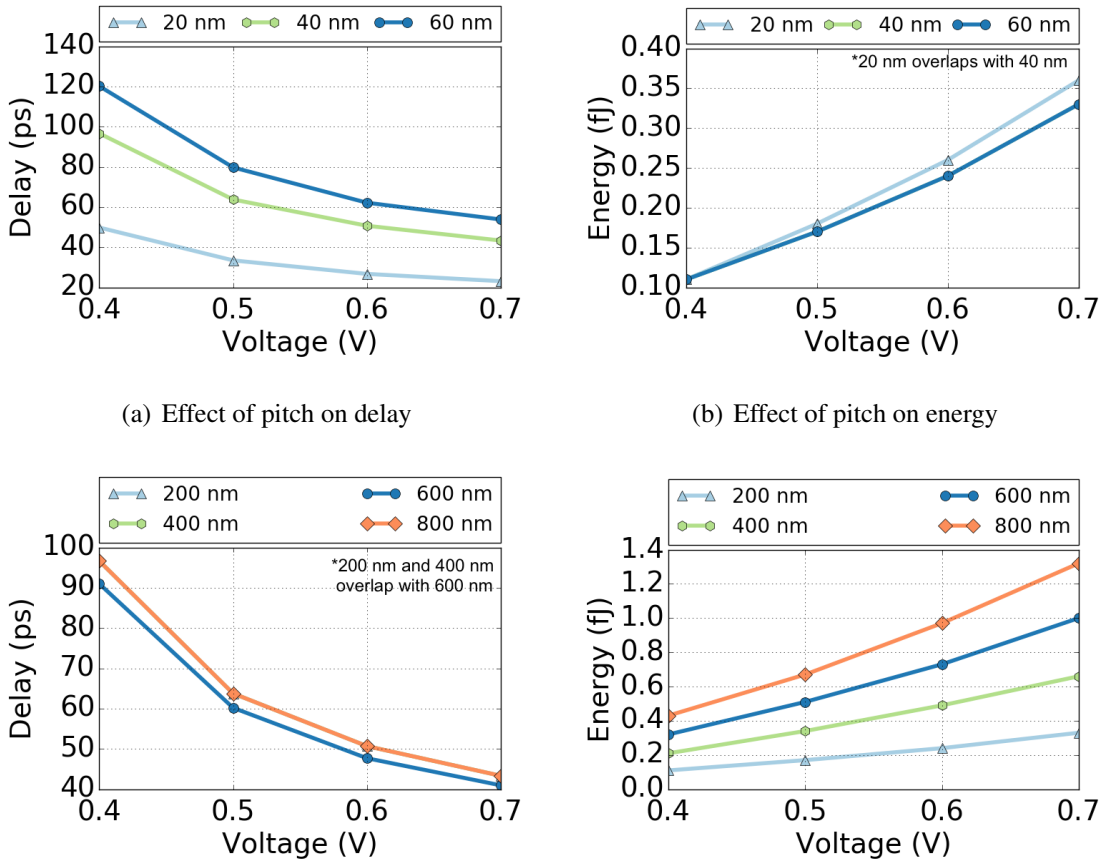
Threshold voltage of the intrinsic CNT channel in a CNFET can be approximated to the half bandgap, E_g, which is an inverse function of the diameter [15]. For a $\pm 10\%$ diameter (1.2 nm) variation, we get a threshold voltage of 0.33-0.39 V. Hence, 0.4 V is selected to be the lower bound of supply voltage scaling in the voltage study.

Simulations are performed using the 16 nm Virtual Source CNFET HSPICE model from Stanford University's Nanoelectronics Group [22]. The model is built on experimental data collected from multiple transistors built on one CNT with varying channel lengths from 3 μm to 15 nm. However, the model assumes CNTs are perfectly aligned, equally spaced and are of a fixed diameter. Hence, to address this, we choose slightly more pessimistic design parameters, as described in the following subsections.

3.5.2 CNFET Design Parameters

The strength of a CNFET is determined by the width of the transistor, W, as well as the CNT pitch, s. While high ρ_{cnt} has been reported in previous work, the control of s $(= 1/\rho_{cnt})$, still remains to be mastered. Lee *et al.* predict that a density of 180 CNTs/μm is required to meet the ITRS targets of off-state and on-state currents at the 5 nm technology node [39].

Considering these features of CNFETs, we study the effects of varying s and W on an FO4 inverter's delay and energy as shown in Figure 3.3. While the delay increases with increasing CNT pitch (s), the energy increases with increasing transistor width (W). We



(a) Effect of pitch on delay

(b) Effect of pitch on energy

(c) Effect of width on delay

(d) Effect of width on energy

Figure 3.3: Varying pitch and width of the CNFET

also see that s has a minimal effect on energy. The decrease in delay from decreasing s is countered by the increase in power due to an increase in the number of CNTs (N_{CNT}). Similarly, increasing W has no effect on delay as the FO4 inverter sees an equivalent increase in its output load capacitance. We choose a pessimistic pitch of 40 nm to incorporate worst case variation of CNT pitch and removal of metallic-CNTs. This pitch value is used for the rest of the CNFETs characterized in this work and is in line with contemporary fabrication techniques.

Further, for ease of area comparison against Si-CMOS transistors, we approximately match the width of the minimum-sized transistor in Si-CMOS to our minimum-sized transistor, *i.e.*, a 4-fin Si-FET of width 240 nm (about 60 nm contributed by each fin) is

matched to a CNFET of width 200 nm, resulting in at least 5 CNTs per minimum sized transistor.

3.5.3 Implementation

Since CNFETs have similar characteristics to Si-FETs, it is fairly straightforward to derive basic CNFET gates from already existing Si-FET gates. Using these gates, we created a CCNT standard cell library to analyze the system-level delay, energy and EDP improvement over Si-CMOS. Similarly, we created a PTL-CNT library of the basic cells required for the ALU and multiplier units. We performed synthesis of the processor using Synopsys Design Compiler and preserved the boundaries around the ALU and multiplier units. These components were separated so that they could be profiled individually. The gate-level netlist obtained from synthesis was then converted into an HSPICE netlist for each unit, using the CCNT and PTL-CNT standard cell libraries. 32-bit versions of an RSA and KSA adder, an ALU and a multiplier were created using this methodology as well. The PTL-CNT versions of these modules were further analyzed and restoring logic was inserted periodically for RSA-based designs and optimally, depending on the varying output capacitance, for KSA and the sparse-tree adder. Each of these building blocks were then evaluated at varying voltages for delay and energy. We compare PTL-CNT results against both CCNT designs as well as Si-CMOS results. Based on both delay and energy numbers, a hybrid design of V-scale was made using PTL-CNT and CCNT modules. We maintain performance and reduce area by using PTL-CNT modules for components along the critical paths of the V-scale pipeline, while using low-energy CCNT modules for the rest of the chip.

3.6 Evaluation

In this section, we evaluate each of our core components implemented in Si-CMOS, CCNT, and PTL-CNT. We then evaluate the overall performance of the V-scale pipeline implemented

with Si-CMOS, CCNT, and hybrid CCNT/PTL-CNT.

3.6.1 Adder Analysis

We begin our analysis by studying a single full adder cell, then build both an RCA and KSA adder. Finally, we analyze an ALU design that leverages a hybrid of RCA and KSA.

3.6.1.1 Full Adder

We compare a 20 transistor PTL-based full adder implementation against a traditional CCNT-based 28 transistor mirror adder [38] as well as its counterpart in Si-CMOS. We designed this 20T full adder to obtain a fast Sum and C_{out} with only two transistors on the critical path, as shown in Figure 3.4. We reduced the load for C_{out} by de-multiplexing the shared part of the circuit with Sum, creating two separate circuits to reduce degeneration during cascading of the full adder for larger blocks, unlike the adder and subtractor built by Ding *et al.* [24].

Figure 3.5 compares the effect of voltage scaling on the three full adder designs. The results show that although the delay trends are similar, our PTL-CNT design clearly dominates in terms of energy, leading to a 7-19× EDP reduction over Si-CMOS in the supply voltage range of 0.7-0.4 V.

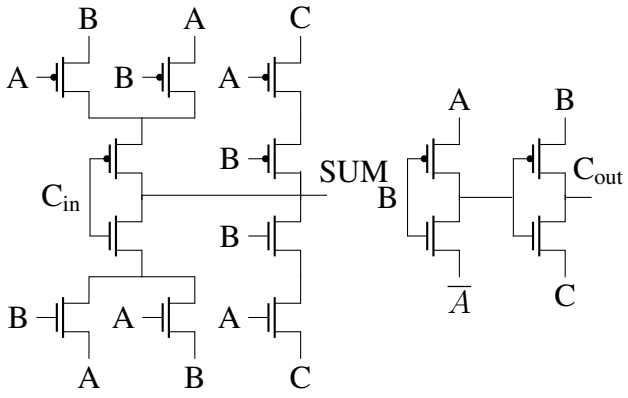


Figure 3.4: Pass transistor-based full adder

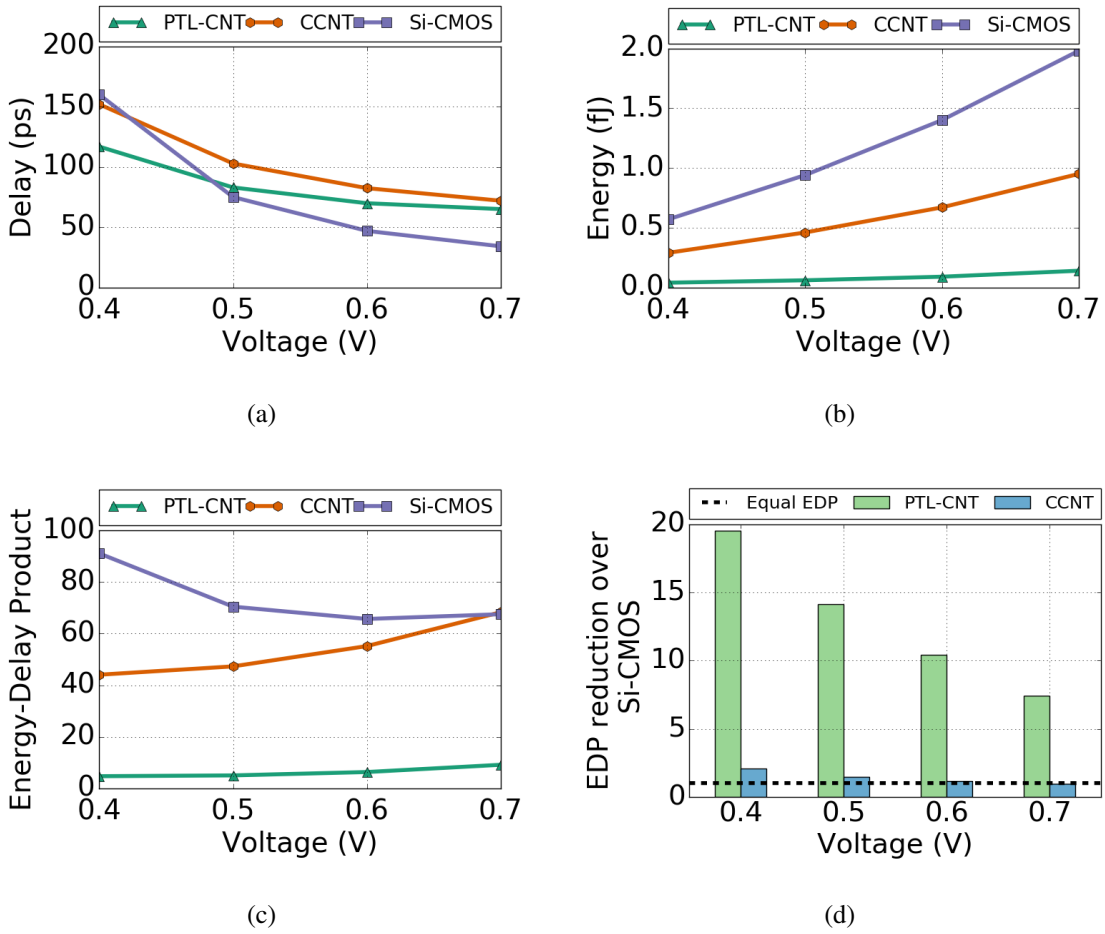


Figure 3.5: Improvement of PTL-CNT and CCNT over silicon for a full adder

3.6.1.2 32-bit Adder and ALU

We implemented an RCA, whose results are shown in Figure 3.6(a) and Table 3.1. In addition, results for the KSA are shown in Figure 3.6(b) and Table 3.2. Our analysis shows that the implementation of a 32-bit RCA using the full adder in PTL-CNT entails a high EDP reduction over the CCNT and Si-CMOS implementations. Although some of the gains seen in the full adder are consumed by the addition of restoring logic placed for PTL. The PTL-CNT KSA implementation saw a smaller improvement in EDP compared to Si-CMOS. This occurred because the KSA required significantly more restoring logic than the RCA, more than offsetting the gains obtained in delay.

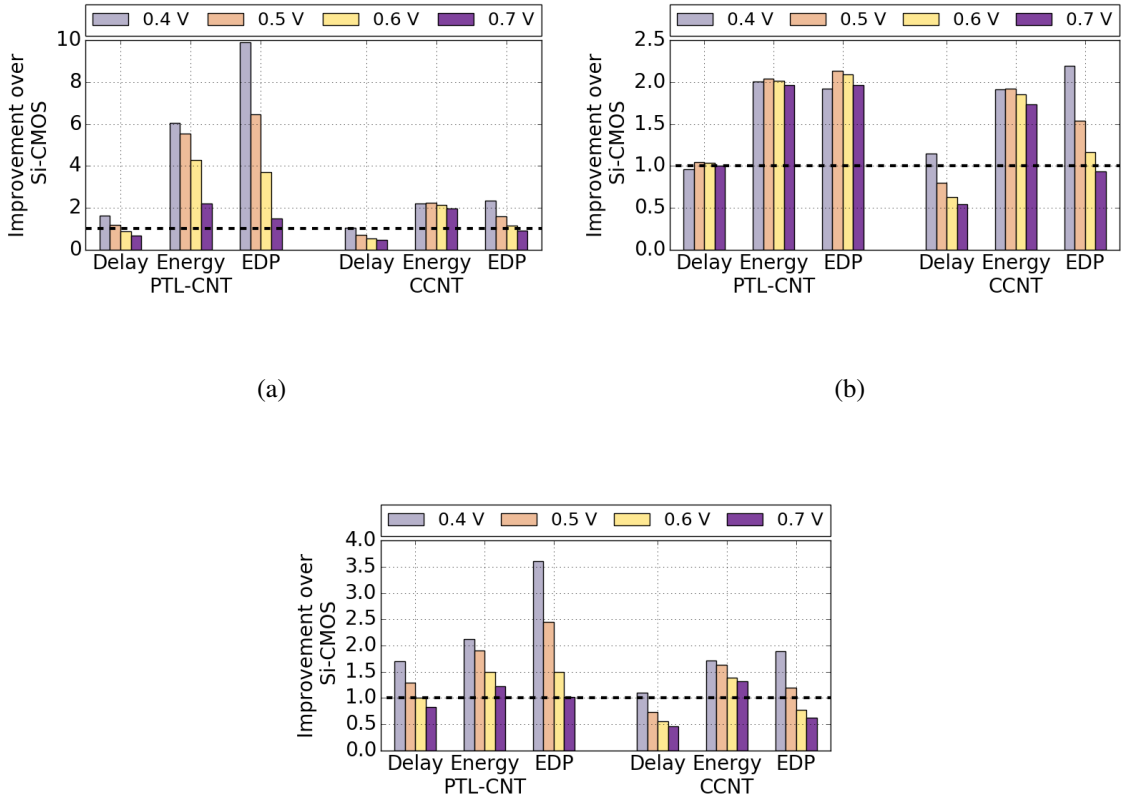


Figure 3.6: Improvement of PTL-CNT and CCNT over silicon for (a) ripple-carry adder, (b) Kogge-Stone adder and (c) V-scale ALU

For the ALU design, we used Synopsys Design Compiler to generate a synthesized netlist. The result, a sparse-tree adder, borrows elements from both KSA and RCA. We implemented a similar sparse-tree adder for our final ALU implementation, to optimize for both area and delay. Figure 3.6(c) and Table 3.3 present the results of the ALU design. We find that the PTL-CNT ALU clearly outperforms the Si-CMOS ALU with an EDP reduction of 3.6× at 0.4 V.

Table 3.1: Ripple-carry adder design results

Volt. (V)	Delay (ns)			Energy (fJ)		
	PTL-CNT	CCNT	Si-CMOS	PTL-CNT	CCNT	Si-CMOS
0.4	1.9	2.9	3.1	2.4	6.5	14.4
0.5	1.2	2.0	1.4	4.2	10.5	23.3
0.6	1.0	1.6	0.9	8.0	16.2	34.5
0.7	1.0	1.4	0.7	22.0	24.4	48.3

Table 3.2: Kogge-Stone adder design results

Volt. (V)	Delay (ns)			Energy (fJ)		
	PTL-CNT	CCNT	Si-CMOS	PTL-CNT	CCNT	Si-CMOS
0.4	1.0	0.8	1.0	4.9	5.1	9.8
0.5	0.4	0.6	0.4	7.7	8.2	15.8
0.6	0.3	0.4	0.3	11.6	12.6	23.4
0.7	0.2	0.4	0.2	16.7	18.9	32.8

3.6.2 Multiplier

Results for the multiplier design are presented in Figure 3.7 and Table 3.4. We find a similar trend at higher voltages. The PTL-CNT multiplier has an EDP gain of 1.6× at 0.4 V, which is less than the 2× of the CCNT multiplier, due to the large overhead of restoring buffer insertion in the PTL-CNT design. Hence, we choose a CCNT-based multiplier for the pipeline design.

Table 3.3: V-scale ALU results

Volt. (V)	Delay (ns)			Energy (fJ)		
	PTL-CNT Hybrid	CCNT	Si-CMOS	PTL-CNT Hybrid	CCNT	Si-CMOS
0.4	2.1	3.2	3.5	20.5	25.4	43.5
0.5	1.2	2.2	1.6	38.3	44.4	72.7
0.6	1.0	1.8	1.0	73.4	79.1	109.6
0.7	0.9	1.5	0.7	127.4	118.5	156.5

Table 3.4: Array multiplier results

Volt. (V)	Delay (ns)			Energy (fJ)		
	PTL-CNT	CCNT	Si-CMOS	PTL-CNT	CCNT	Si-CMOS
0.4	3.7	2.2	2.8	276.2	293.4	560.6
0.5	1.9	1.5	1.2	429.1	470.6	906.9
0.6	1.4	1.2	0.7	610.0	728.1	1351.4
0.7	1.1	1.1	0.5	930.7	1071.9	1902.7

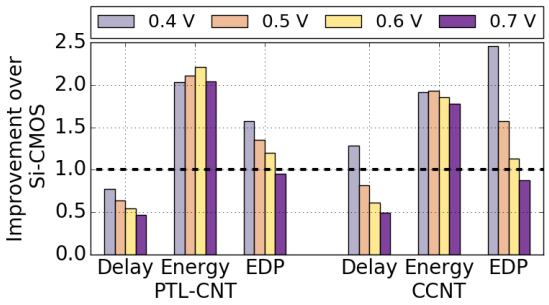


Figure 3.7: Improvement of PTL-CNT and CCNT over silicon for the multiplier

3.6.3 Registers

Since a D-flip flop mostly consists of inverters and transmission gates, we only build Si-CMOS and CCNT-based implementations. Though Si-CMOS performs better than CCNT flip flops by a small margin at higher voltages, the CCNT flip flop wins back at 0.4 V with an EDP gain of 1.8× as shown in Figure 3.8.

3.6.4 Full Pipeline

Figure 3.9 and Table 3.5 present the results of our full RISC-V pipeline design. We find that the V-scale core built using CCNT shows a 1.0-2.9× improvement in EDP over a Si-CMOS based core for a supply voltage range of 0.7-0.4 V. To improve this further, we analyzed the critical path and found that the ALU and parts of the multiplier were on the critical path. For that reason, we constructed a V-scale pipeline with the PTL-CNT versions of the ALU components. We obtained a 2-5× reduction of EDP over Si-CMOS with this implementation, which is also a 1.7-2× improvement over the entirely CCNT design. The

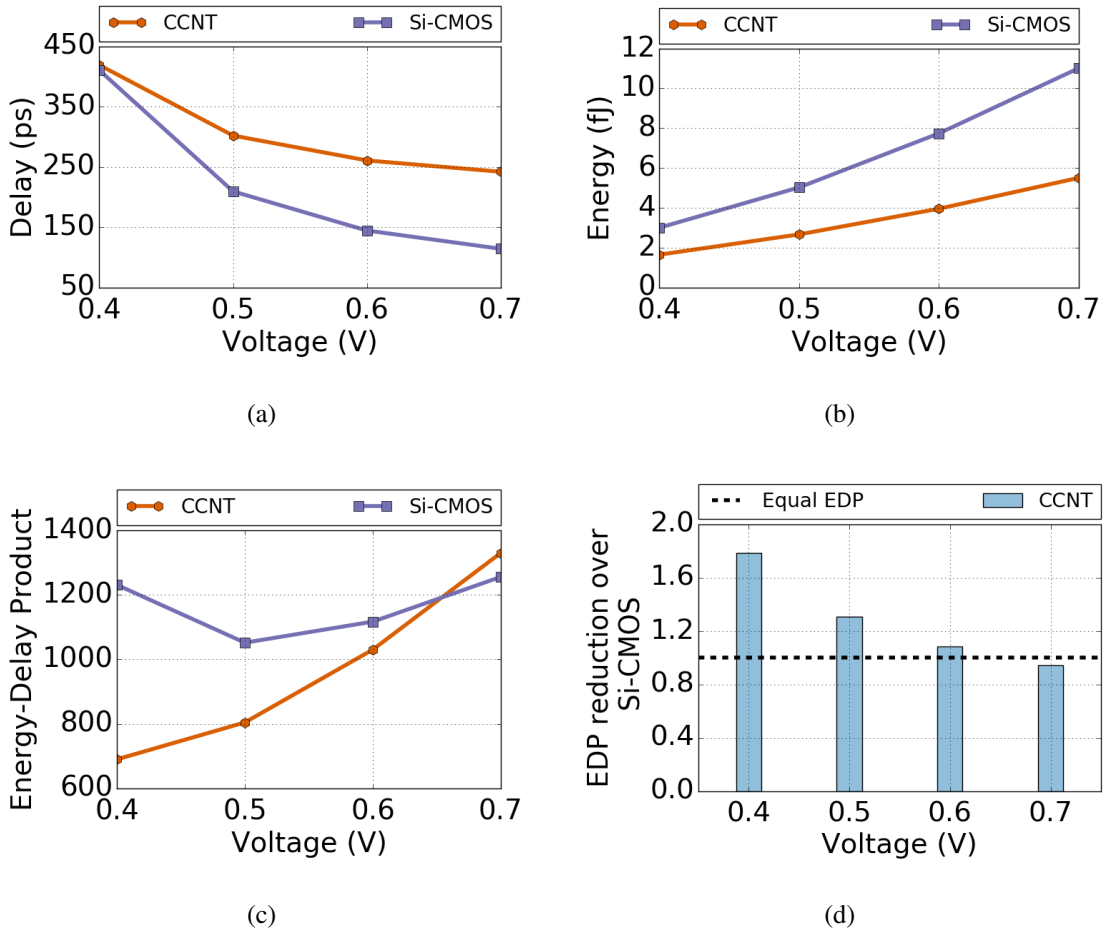


Figure 3.8: Improvement of CCNT over silicon for the D-Flip Flop

results clearly show that CNFETs are a better fit for low voltage and energy-efficient designs, and that judicial use of PTL can greatly improve the effectiveness of CNTs.

While the individual components show on average a $\sim 2\times$ improvement in EDP, the overall CPU pipeline shows a $5\times$ improvement. This happens because the analysis for individual components were done at the maximum frequency for those components. When integrated into the entire pipeline, the critical path is comparatively longer than the propagation time of each individual component on it, and hence those units only contribute leakage power to the system's power for rest of the clock cycle. Since Si-CMOS has a larger penalty for leakage than CNFETs, this compounds to produce the $5\times$ improvement. We also achieve a 17% reduction in area of the hybrid pipeline in comparison to the Si-CMOS configuration.

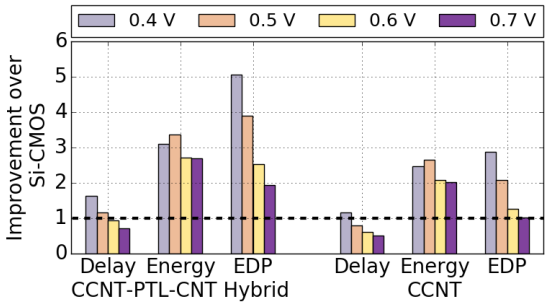


Figure 3.9: Improvement of CCNT-PTL-CNT Hybrid and CCNT over silicon for the V-scale pipeline

Table 3.5: V-scale pipeline results

Volt. (V)	Delay (ns)			Energy (fJ)		
	PTL-CNT Hybrid	CCNT	Si-CMOS	PTL-CNT Hybrid	CCNT	Si-CMOS
0.4	3.0	4.2	4.9	508.6	639.8	1578.0
0.5	1.9	2.8	2.2	747.6	947.9	2511.6
0.6	1.5	2.3	1.4	1044.2	1356.3	2832.0
0.7	1.4	2.0	1.0	1430.2	1908.4	3863.0

3.7 Conclusions

Although many breakthrough fabrication techniques to synthesize carbon nanotubes have been invented, we still need circuit and architectural overhauls along with further fabrication improvements to suit CNFETs while building larger blocks and systems to gravitate their capabilities. Considering the low threshold voltage, low power dissipation and equal PFET and NFET strength of carbon nanotubes, we built a RISC-V pipeline using pass transistor logic-based CNT building blocks. We report the energy, delay and EDP of these smaller logic blocks and build a whole pipeline using a hybrid of pass-transistor logic and complementary logic for complex modules of the pipeline. The results clearly show that CNFETs are a better fit for low-voltage and low-power designs. While individual blocks show up to 3.6× improvement in EDP compared to 16 nm Si-CMOS based designs, the RISC-V V-scale pipeline shows an EDP improvement of 5×, bringing us one step closer to the full potential of CNFETs.

CHAPTER 4

A Design Framework for High-Variation Transistor Technology

With the gradual slowdown of Moore's law, the semiconductor industry has seen the emergence of various new technologies to supplement or replace silicon-based transistors. Although physical scaling of silicon (Si) is predicted to end with the 5 nm node [7], there is no alternative technology at the moment that has the capability to match the yield and performance of Si for existing designs. Extensive research on both the manufacturing as well as architecture fronts are required to move innovation forward to create large scale chips using these new technologies. However, aggressive manufacturing research will not be done unless a product is marketed and products cannot be developed profitably because of the high variation in the manufacturing process, leading to a viscous cycle. Hence, to break this causality dilemma, we as architects need to develop design flows for high variation technologies that can overcome the reliability concerns of a new technology and compete with the highly optimized state-of-the-art Si-based designs.

Carbon Nanotube Field Effect Transistor (CNFET) is a promising alternative to Si-based devices due to its exceptional electrical, thermal and transport properties, such as high carrier mobility, smaller gate capacitance and better current endurance [40]. Having an order of magnitude better energy-delay product (EDP) compared to Si CMOS logic, CNFET can be used to build highly energy-efficient integrated circuits [41]. CNFETs were also demonstrated in Monolithic 3D-ICs by building CNFETs with Si CMOS gates [40].

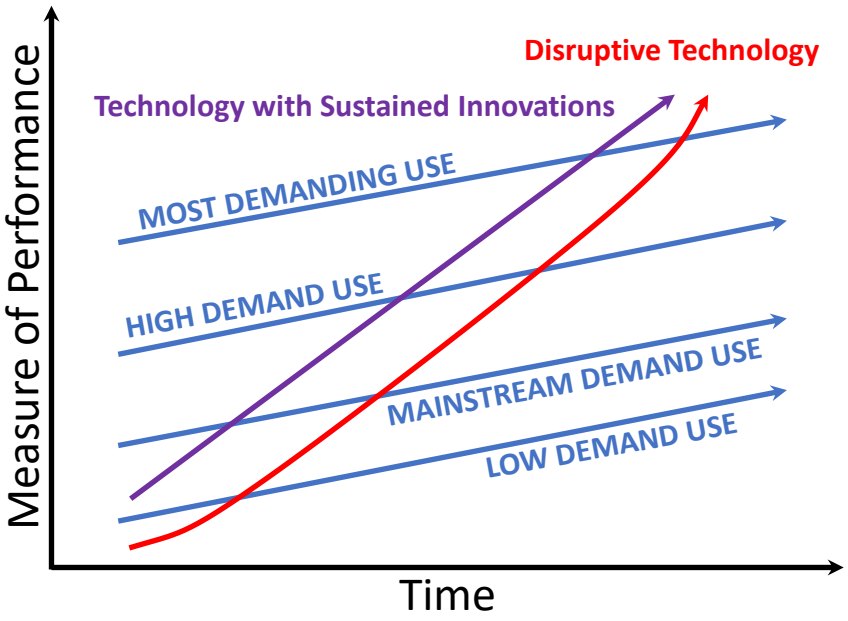


Figure 4.1: Path to sustenance for a disruptive technology [1]

Recently, CNFETs have also been adopted in the design and development of next-generation of 3D Monolithic System-on-a-Chip technology to create densely integrated logic and memory products [42].

However, the current CNFET manufacturing process is riddled with imperfections [43]. Chemical synthesis of CNTs does not provide precise control over the locations of individual CNTs on the wafer and consequently, significant variations can exist in the spacing between CNTs. This non-uniformity, which is expressed as *CNT count variation*, leads directly to spatial non-uniformity in the electronic properties of CNFETs, resulting in increased delay, signal level attenuation and failure. Moreover, a single defective CNFET can cause faults on the gate level, such as higher leakage, less balanced rise/fall delays or too much driver strength for either pull-up/pull-down path [44]. These problems exist because the technology itself, albeit promising, is still developing from infancy.

As shown in Figure 4.1, for any disruptive technology to sustain growth and innovation, it usually requires a low-end market with a low barrier of entry to gain initial sources of revenue. With the initial investment, the technology can then improve to enter the mainstream markets and eventually outperform existing technologies. For example, as

observed by Christensen in [1], flash memory, a disruptive technology, costed 5-50× more than the hard disk per megabyte of memory and was not as robust for writing. However, flash chips achieve higher performance and reliability at lower power. Flash memory started in small value networks, such as digital cameras and modems. Comparatively, disk drives are too big, fragile and power consuming for these applications. After flash chips succeeded as a niche, the industry began marketing specialized storage systems in portable packages, like the thumb drive. Today, Solid State Drives, made using flash memory, comprise the fastest growing segment of the storage, arriving to this stage because of the initial investments in low-end markets.

Hence, to sustain the development of CNFET designs, it is necessary to introduce architectural and circuit innovations that improve yield and help create designs for the low-demand market, as shown in Figure 4.1. These solutions will help drive the demand for CNFET products, increasing the number of products manufactured. The initial income obtained can be leveraged by foundries to improve the process technology and expand future leading to a stable CNFET market.

To achieve this goal, we first develop a flexible variation model based on CNT density fluctuations, that allows the designer to mimic the yield obtained for different manufacturing processes. Leveraging the fact that CNFETs are CMOS compatible, we built a 16 nm CNFET standard cell library and characterized it for voltages varying from 0.4 V to 0.7 V, creating a design library that can be used to synthesize logical designs. Furthermore, to enable the commercialization of CNFET-based products, we propose the use of a multi-granular reconfigurable 3D architecture, **3DTUBE**, which improves yield and throughput of high-variation designs. We show that for a failure rate of 10 ppm, the pipeline-stage level and module-level solution achieves 2.0× and 2.5× improvement in performance over a baseline 8-core OpenSPARC T1 processor with no reliability solution, respectively. We also show by employing 3DTUBE for the current CNT process, we can achieve up to 6.0x higher throughput and 3.1x lower Energy-Delay Product (EDP) than that of a 16 nm Silicon-based

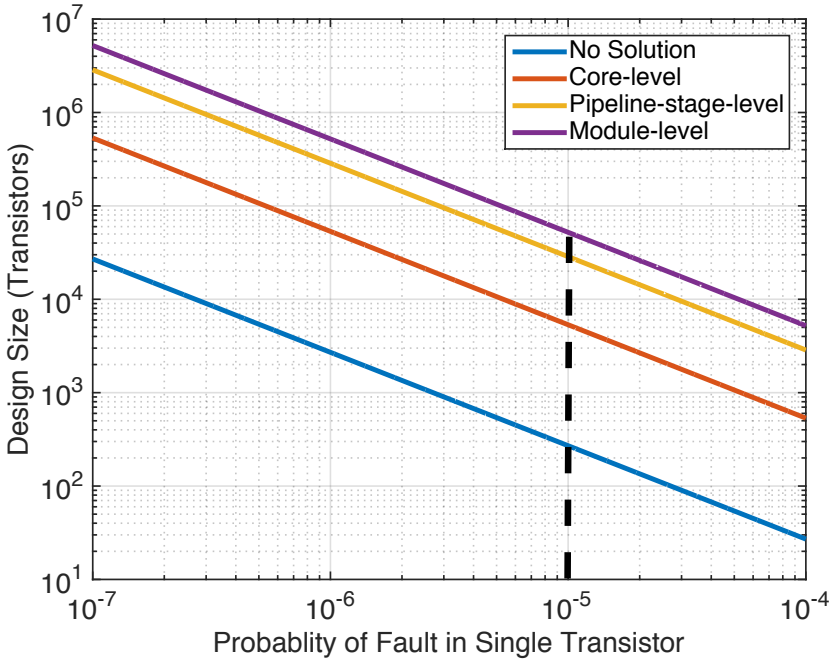


Figure 4.2: Compute size achieved for 99.73% yield if no reliability, core-level, pipeline-stage level, or module-level solution is employed for varying fault rates. Dashed line denotes the current process's failure rate of a CNFET (10 ppm).

design at a minimal area cost of 7.4% and frequency overhead of 8.2% over an unprotected multi-core design.

4.1 Motivation

Deploying technology in a large-scale production requires a high design yield of 3-sigma or above for the product to be profitable. However, due to the infancy of manufacturing processes of emerging technologies, we cannot achieve this yield without compromising on size of design.

High variation observed in the manufacturing process [8] is a significant challenge to be conquered before the benefits of CNFETs can be reaped. Manufacturing imperfections leading to CNT count variation can affect CNFET performance and reliability. Despite the strides that have been made in CNFET manufacturing process over the years, the latest CNFET process demonstrates a high failure rate of 10 ppm (parts per million) [9]. As seen

in Figure 4.2, for a technology with a failure rate of 10 ppm to achieve a yield of 99.73% (3-sigma process), it can only realize a maximum design size of 300 transistors, which is negligible in comparison to the 800 million transistors in the Intel Core i7 processor.

Deriving motivation from the row/column redundancy utilized in SRAMs [31], we observe that by adding redundancy to the design, we can overcome the challenge of low yield and realize larger designs. Adding a redundant second core improves the design size by 20× as shown in Figure 4.2. Further, adding redundancy at the pipeline-stage and module level improves the size by 190× and 255×, respectively, over the no solution design. This allows us to design small reliable CNT processors for the low-end market. However, naively adding redundant components adds expensive overheads.

Gaining insight from these results and using the ability of CNFETs to be integrated into monolithic 3D circuits [40], we later demonstrate in Section 4.3, an efficient multi-core 3D architecture that can be reconfigured at multiple granularities to provide an inherent redundancy that improves the yield and performance of CNFET designs at very low overheads.

4.2 Variation Model

Recent fabrication techniques have helped make large scale CNFET manufacturing processes possible. However, the CNFET manufacturing is imperfect, due to the presence of metallic carbon nanotubes (m-CNTs), imperfect m-CNT removal processes, chirality drift, CNT doping variations in the source/drain extension regions, diameter variations and density fluctuations due to non-uniform inter-CNT spacing [43]. Based on prior work [29], variation in CNT count caused due to CNT density variations and m-CNT-induced variations is the major contributor of delay degradation, up to 60% in CNFET circuits at the 16 nm technology node [29]. The lack of precise growth or placement of CNT on a wafer along with the removal of m-CNTs lead to high variation in the CNT density. Furthermore, 3DTUBE can be used to countermeasure any category of CNFET variation. Therefore, for

the purposes of this work, we build a flexible variation model based on the fluctuations in the number of CNT and analyze its impact on the yield and performance of CNFETs.

4.2.1 CNT Distribution

To model the CNT distribution observed in CNFETs and due to its prevalence in previous studies [9, 27, 44], we adopted a Gaussian distribution function. Certain works have also modeled the CNT count distribution as a Weibull distribution [28] or mixed joint distribution [26]. We defer the evaluation of these distributions to a future work. Our model uses the distribution in the inter-CNT spacing, CNT pitch (s), to obtain the variation in number of CNTs (N). Hence, the distribution of N can be constructed using:

$$(\mu_N, \sigma_N^2) = (W/\mu_s, W\sigma_s^2/\mu_s^3) \tag{4.1}$$

where μ_N is the mean of N, σ_N is the standard deviation of N, W is the width of a CNFET, μ_s is the mean of CNT pitch and σ_s is standard deviation of pitch.

Based on the failure rate of 10 ppm (probability of no CNTs using W = 2um, N = 9, σ_N = 2.1) obtained from [9], for a minimum width transistor of 100 nm, we chose a μ_s of 20 nm and a σ_s of 10nm for our baseline model and consider a yield failure to occur when $N \leq 0.25$ CNTs. These parameters can be changed for improvements in the manufacturing process.

4.2.1.1 Effect of CNT Distribution on Yield Failures

Yield failures occur when a transistor has a negligible number of CNTs making it non-functional. To observe the effect of the distribution of CNTs in a device on yield failures, we vary the standard deviation of the pitch from 5 nm to 10 nm (current process technology) as shown in Figure 4.3. This trend shows that a small reduction in the standard deviation of the CNT pitch can reduce yield failures significantly.

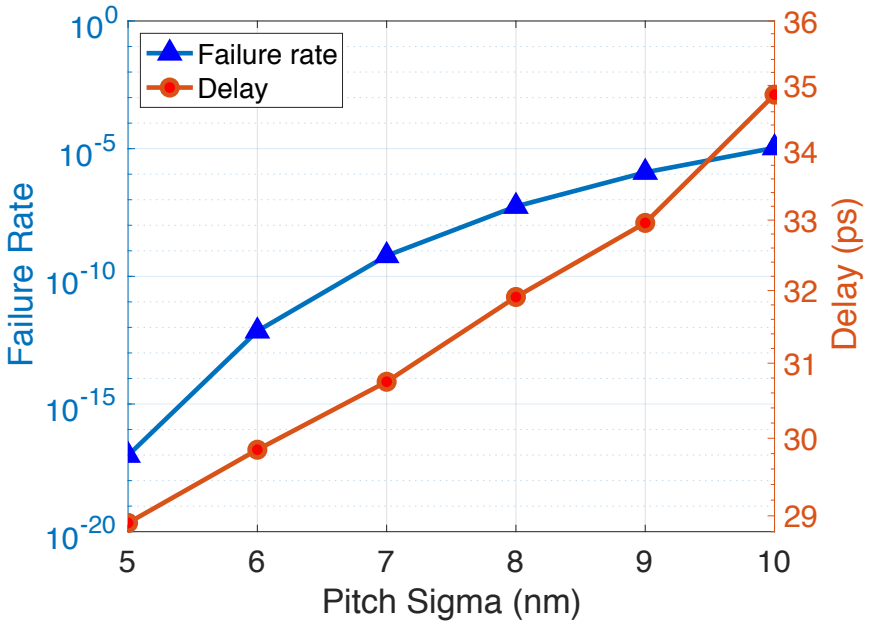


Figure 4.3: Effect of CNT distribution (varying pitch sigma) on FO4 inverter 3σ delay and yield failures for W = 100 nm and s = 20 nm

This analysis is used in Section 4.4 to decide the granularity of our solution. For a high failure rate process, we require redundancy at a fine granularity to create large-scale designs.

4.2.1.2 Effect of CNT Distribution on Delay

The distribution of CNT can affect the strength of a transistor, which in turn affects the *delay of a gate* and *performance of a chip*. As shown in Figure 4.3, with increasing standard deviation of pitch, an FO4 (fan-out of 4) inverter's 3σ delay is affected *almost linearly*, implying that while the transistor yield rate can have a higher effect on the throughput of a processor, delay variation affects the clock period and timing-based optimization.

4.2.2 Variation Suite

When creating huge designs, it is inefficient to obtain the critical paths and find the derate in frequency due to variation in the process technology. Hence, we automate the process by generating a variation suite, which contains the percentage variation in delay for a path of length l, and average drive strength d. Note that the creation of this suite is a one-time

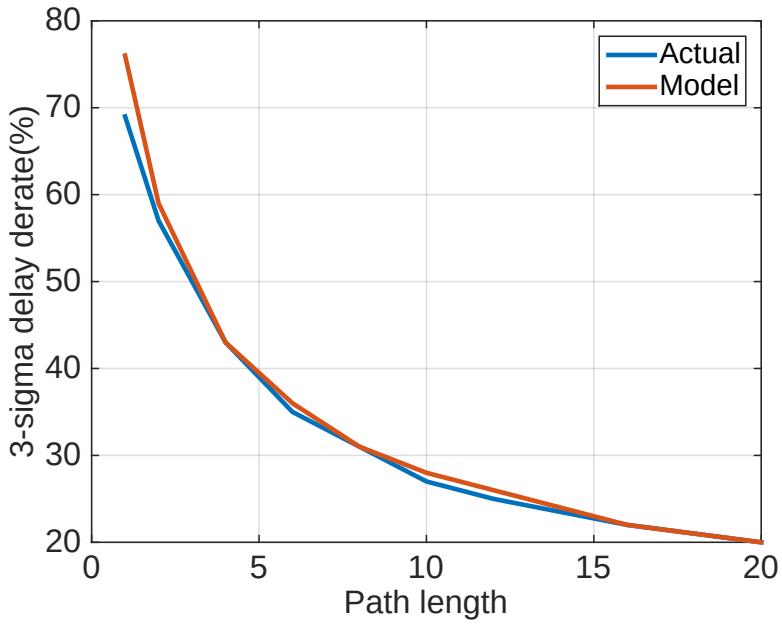


Figure 4.4: 3σ-delay derate of statistical model in comparison to actual derate for FO4 inverter chains, of drive strength 1, of varying lengths from 1 to 20.

process for any technology node.

The derate value for a path of length, l, and average drive strength, d is calculated as $3\sigma_{l,d}/\mu_{l,d}$ where, $\mu_{l,d}$ and $\sigma_{l,d}$ are the mean and standard deviation of delay obtained from a chain of l FO4 inverters of strength d. It has been shown that the delay derate reduces with longer paths and larger gates [45]. Hence, by approximating the original path with the variation seen in inverters, we are slightly pessimistic with our model.

Furthermore, to save design time spent on generating Monte Carlo results for various combinations of depth and strength, we use a statistical model used in static timing tools [46] to estimate the standard deviation for long inverter chains ($l \geq 5$). We estimate the model based on the following equations:

$$\mu_{l,d} = l.\mu_d \tag{4.2}$$

$$\sigma_{l,d} = \sigma_d.\sqrt{l.(1 + 2\rho_d(1 - \frac{1}{l}))} \tag{4.3}$$

where, $\mu_{l,d}$ and $\sigma_{l,d}$ are the mean and standard deviation of delay for a path of length l

and strength d, μ_d and σ_d are the mean and standard deviation of delay of an FO4 inverter with drive strength d and ρ_d is the correlation between two adjacent FO4 inverters of strength d, due to *slew and load capacitance dependence*. On evaluating our statistical model using Monte Carlo simulations, we found that the model's variation estimate is within 2% that of the 3σ delay variation noticed in the original path for $l \geq 5$ as show in Figure 4.4.

4.3 Architecture

Recent reliability solutions propose to break apart the hardware units of classic hard-wired pipelines, dissolving them into a sea of redundant hardware components [47, 48]. Upon fault detection, these designs can reconfigure the hardware by replacing faulty components with new ones. These fine-grained reconfigurable and fault isolating systems can maintain reliability in the presence of faults without sacrificing performance for 2D and 3D systems. Furthermore, leveraging the feasibility of building monolithic CNFET 3D-ICs [40], we adopt the solution, 3DFAR proposed by Bagherzadeh and Bertacco [48] to build our solution, 3DTUBE.

3DTUBE is a novel multi-granular, 3D reconfigurable reliability solution for CNT-based processor designs, which leverages the system's natural redundancy to provide robustness against permanent failures. Each manufactured chip can be configured to route instructions through functioning components and *detour around failed pipeline stages or modules* based on the failure pattern seen in that particular chip, unlike traditional architectures that execute instructions on fixed paths.

As shown in Figure 4.5, by replacing the direct connections at the boundary of each pipeline-stage or module with interconnect switches, we create a network of resources in which each component is connected to all instances of the subsequent stage. To minimize the performance loss from inter-stage communications, we use multiplexer-based full crossbar switches because of their non-blocking access to all their inputs, and small number of input

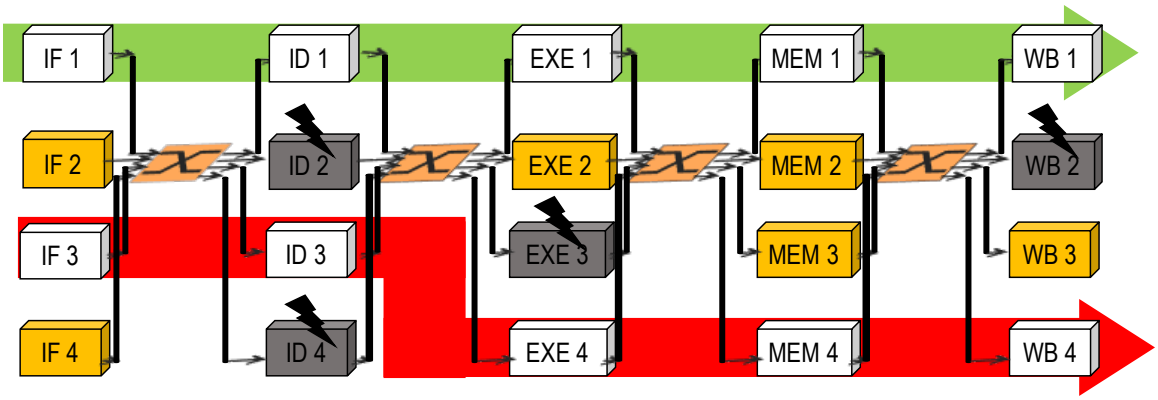


Figure 4.5: Schematic of 3DTUBE, where identical components are stacked vertically, and crossbars are inserted between stages. In this four-failure scenario, a regular 3D CMP would only have one working core. In contrast, 3DTUBE dynamically reconfigures to connect healthy units creating 2 complete cores (red and green stripes). Stages in orange are idle units that could not be used to form functional pipelines.

and output ports that are not prohibitively expensive. By adaptively routing around failed stages, we can salvage working units to effectively repair the system by creating logical cores across the 3D structure to tolerate variation-based failures.

Based on the manufacturing defects (yield or timing failure) detected in a chip, the victim unit (*i.e.*, a pipeline stage or module) is isolated and an identical unit from another layer of the 3D fabric, is used for execution. Hence, the logical core may comprise of elements from various vertical layers. With reference to the example in Figure 4.5, in which 4 defects have disabled units on different vertical layers, our architecture can build two complete cores dynamically (red and green stripes), while a traditional solution (2D or 3D) would have only one.

Figure 4.6 shows two ways in which the pipeline and the crossbar can be divided to create a module-level architecture; *parallel* or *serial decomposition*. In parallel decomposition (Figure 4.6.b), the logic and crossbar in each pipeline stage is divided into two parallel parts. In this case no area or delay overhead associated with the crossbars is added compared to the pipeline-level solution (Figure 4.6.a), except new set of control signals for the parallel MUXes. In serial decomposition (Figure 4.6.c), the logic and crossbar in each pipeline stage is divided into sequential parts. The area and delay overhead of the crossbars and vertical

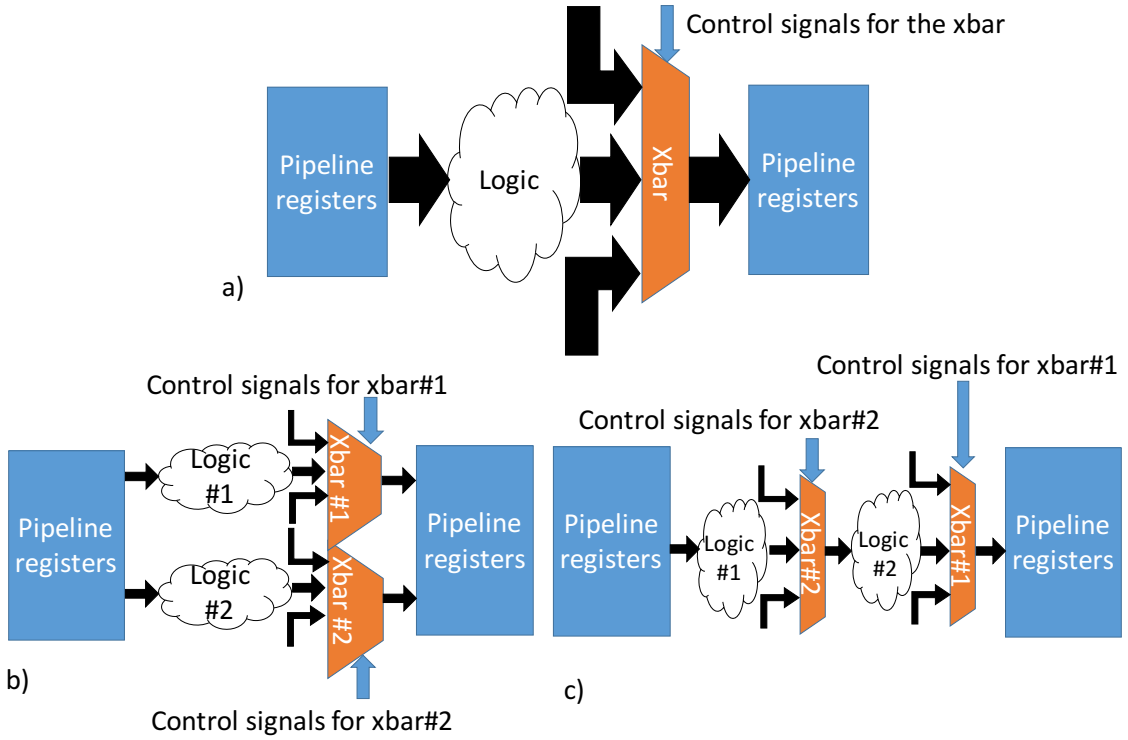


Figure 4.6: a) 3DTUBE pipeline-level structure b) Parallel module-level architecture c) Serial module-level architecture

connections for n serial modules would be n times that of the pipeline-stage level solution along with the new set of control signals for n different MUXes. Hence, we choose parallel decomposition as our preferred approach to create the module-level solution. Depending on the defect rate identified, we can adjust the granularity to modular level, pipeline-level or core level obtaining higher yield.

Similar to 3DFAR, 3DTUBE's cross-layer interconnect switches do not require any buffering [48]. Propagation delays on vertical Through Silicon Vias (TSVs) are minimal (more than an order of magnitude faster than in conventional 2D layouts) due to the much shorter lengths to be traversed. Furthermore, to account for manufacturing issues associated with a monolithic 3D-IC, if any of the switching MUX logic fails, the module or pipeline associated with it (component whose input is the MUX's output) in that specific layer would be unusable, but similar stages in other layers, or other stages in that layer would still be functional and can be used to create a logical core. If all MUXes fail, then it would lead to a

failure of the entire system. However, as the MUX area is a small percentage of the design compared to the stages, the probability of losing a MUX before the stage it is connected to is negligible. Similarly, reliability is also important when using MIVs (or TSVs), as the failure of a single MIV may cause system failures. Yield and reliability improvements for these MIVs are achieved through a range of redundancy techniques and sparring along with several diagnosis and repair mechanisms [49]. For our design, we account one spare MIV for every 100 MIVs.

4.4 Design Flow for High-variation Technology

This section describes the various design flow steps for a high CNT-density variation technology to be able to produce large scale designs with high yield and performance.

4.4.1 Standard Cell Library

The first step in any design flow is to create the basic building blocks, a standard cell library. CNFETs use carbon nanotubes as the channel medium between the source and the drain, instead of silicon. Leveraging the fact that the behavior of a CNFET is similar to a Si-FET, *i.e.*, we observe a linear region followed by a saturation region in the drain current, I_{DS}, as a function of increasing gate-source voltage, V_{GS} [13], we derive CNFET gates from an already existing 16 nm Si-finFET cell library to create a 16 nm CNT-based standard cell library. To do so, we match a minimum width 16 nm transistor in the Si-based library to a 100 nm width and 20 nm pitch CNFET, *i.e.*, a minimum width CNFET has 5 CNTs.

4.4.2 Library Generation

Using the 16 nm CNFET standard cell library, Stanford's CNFET Verilog-A model [22] and Cadence's library characterization tool, Liberate, we generate a design library required to synthesize designs in Synopsys's Design Compiler for operating voltages varying from

Table 4.1: Synthesis results of an OpenSPARC T1 core optimized for delay

Volt. (V)	Delay (ns)		Energy (pJ)		EDP reduction
	CNT	Si	CNT	Si	CNT over Si
0.4	0.65	5.00	7.00	3.97	4.36
0.5	0.50	1.50	9.97	8.65	2.60
0.6	0.40	1.00	14.40	12.81	2.22
0.7	0.35	0.45	20.23	19.87	1.26

0.4 V to 0.7 V. For a fair comparison of CNT designs generated using this flow, we also generate the design library for the Si-based 16 nm standard cell library.

We demonstrate the results obtained from synthesizing a SPARC-ISA based OpenSPARC T1 in-order CPU core, with no variation, using both the CNT-based and Si-based design libraries for varying voltages in Table 4.1.

4.4.3 Design Methodology

We use Stanford's CNFET Verilog-A model [22] to create the 16 nm CNFET standard cell library from a 16 nm Si-based finFET standard cell library as described in Section 4.4.1. Using this CNFET cell library and the Verilog-A model, we generate a design library. We also add the variation model to the Verilog-A model as described in Section 4.2. We can use this variation equipped model for both yield and delay analysis of designs. We then generate the variation-suite for path lengths 1 - 5 and drive strength from 1 to 32 using 3-sigma Monte Carlo simulation. For longer path lengths, we use the statistical model to derive an approximate 3-sigma delay variation.

Next, we synthesize an RTL design using the design library to generate a timing report and obtain critical paths to be processed by the variation suite to obtain derated paths (timing of paths adjusted with delay variation). The derated paths and the yield rate of the process are used to derive the granularity at which 3DTUBE must operate to optimize for design yield and performance. For failure rates greater than 1.6×10^{-6}, we would build a module-level system, and failure rate less than that, a pipeline-stage-level 3DTUBE.

4.5 Evaluation

This section evaluates the usage of 3DTUBE and compares its performance at multiple granularities against baseline 3D CNFET and Si designs.

4.5.1 Performance and EDP Analysis

We evaluate 3DTUBE on an OpenSPARC T1 processor which implements the 64-bit SPARC V9 architecture [50]. The OpenSPARC T1 processor contains 8 in-order, five-stage pipelined, single-threaded cores. Each SPARC core has a 16 KB L1 instruction cache, an 8 KB data cache, and fully-associative Instruction and Data Translation Look-aside Buffers (TLB). The 8 cores are connected through a crossbar to an on-chip unified 3 MB L2 cache. This processor can achieve an IPC of 1.68 for the SPECJBB 2005 benchmark [50]. The physical design of each core comprises of a total of 80k transistors.

We evaluate the pipeline-level and module-level solutions for operating voltages varying from 0.4 V to 0.7 V by considering two baselines; one, a Si-based design of the OpenSPARC T1 processor evaluated at 1 ppb transistor failure rate and two, a CNT-based design of the processor with no reliability solution. The resulting throughput of our solutions in comparison to the two baselines for differing failure rates can be seen in Figure 4.7. We show the comparison of EDP for two granularities evaluated at 10 ppm over the Si-based baseline at 1 ppb in Table 4.2.

Table 4.2: EDP reduction of 3DTUBE for an 8-core design at 10 ppm failure rate over the Si-based design evaluated at 1 ppb failure rate

Volt. (V)	EDP Reduction of 3DTUBE over Si	
	Pipeline-level	Module-level
0.4	2.5	3.1
0.5	1.5	1.9
0.6	1.3	1.6
0.7	0.7	0.9

At 0.7 V, for a transistor failure rate of 10 ppm, the module-level design achieves

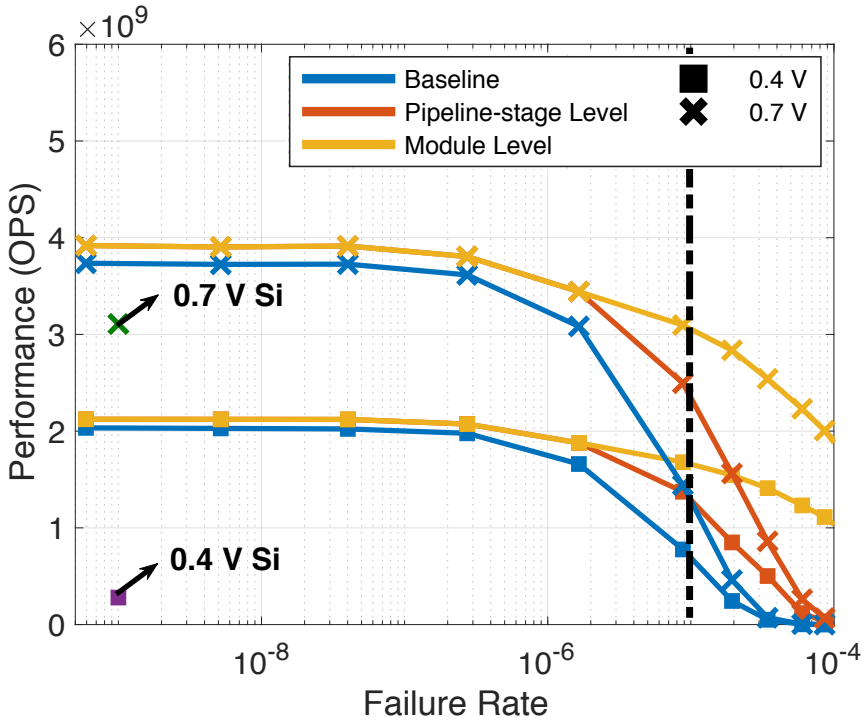


Figure 4.7: Performance of baseline, pipeline-stage level, and module level solutions of 3DTUBE for varying transistor failure rate. Dashed line denotes the current process's failure rate of a CNFET (10 ppm). Si-based design is evaluated at a failure rate of 1 ppb.

a throughput of 3.1 GOPS (Giga operations per second), pipeline-stage design achieves 2.4 GOPS while the CNT-based baseline achieves 1.3 GOPS throughput for the SPECJBB benchmark, *i.e.*, by employing the module-level technique, we achieve 2.5× higher throughput over the CNT-based baseline. Although, at 0.4 V, the module-level 3DTUBE design achieves a 6.0× higher throughput and 3.1× lower EDP, at 0.7 V, it achieves the same throughput at 12% higher EDP compared to the silicon-based baseline.

As shown in the Figure 4.7, we can use the module-level optimization for failure rates above 1.6 ppm and the pipeline-stage level optimization to improve throughput for failure rates above 0.1 ppm. For failure rates below 0.1 ppm, the performance of all three designs saturates due to the size of OpenSPARC T1 design. The range of failure rate for the deployment of either levels is highly dependent on the size of the design and is determined during design flow.

4.5.2 Frequency and Area Overhead

To evaluate our architecture, based on detailed measurements done on a physical design, we analyzed the propagation delays for two layouts of a 4-core OpenSPARC T1 processor: first, a 2D layout with the 4 cores in a 2x2 formation with switches placed at the center of the formation and second, a 3D layout with the 4 cores stacked above each other as shown in Figure 4.5. The worst-case propagation delay for signals going from the output of one stage, through an interconnect switch and to the input of the next stage, for the 2D layout is 20× higher when compared to the 3D layout. This vast difference is due to the much shorter distances that must be traversed to reach a corresponding unit in the three-dimensional solution. Based on these findings, our interconnect switch designs and checkers only add an 8.2% frequency overhead. Furthermore, the pipeline-stage-level framework incurs an overhead of 7.4% in area. The area overhead contains the area of crossbars between stages and TSVs to route the signals across 3D layers. Since propagation delays of the vertical connections are low due to the much shorter lengths to be traversed, we can avoid buffering by accommodating a small increase in clock period for TSVs and MUX-based crossbars in our design. The only real overhead comes with the extra control logic needed as the number of layers increases. For parallel decomposition-based module-level 3DTUBE, the only area and delay overhead in addition to the pipeline-stage level solution is generated from new set of control signals which is acceptable for a small number of modules in each stage.

4.6 Related Work

Leveraging the availability of theoretical CNFET models, prior works have conducted extensive studies on the impact of CNFET variations on yield and performance [8, 25]. In particular, they focused on the variation associated with CNT count in a transistor [26, 27, 28], which was identified as the major contributor to the delay variation in CNFETs [29]. Hence, to evaluate the effect of our variation tolerant design framework, we also base our variation

model on the variation associated with number of CNTs. Moreover, our model is flexible to consider future improvements in the process.

Prior works have also shown improvement in the yield of the process at the layout or circuit level [9, 23, 27]. Stanford's Robust System Group also released a variation-aware Design Kit [30]. However, these lower-level solutions target a particular cause of variation and are not generic in nature. Even though we focus on CNT density variations in this work, by optimizing and enhancing yield at the system level, our solution, 3DTUBE, can be used to model and countermeasure any category of CNFET variation or reliability problem. Furthermore, our solution can be employed in conjunction with the circuit-level and layout-level solutions to improve yield and performance. Moreover, our work also generates a design library, that can be used to create large designs quickly while optimizing for the effects of variation seen at the system level.

Many reliability solutions exist for Si-based processors, but they are created for failure rates in the order of 1 ppb (parts per billion) [47, 48]. Scaling these solutions for a high-variation technology like CNFET, can lead to expensive overheads. Although, 3DTUBE borrows its ideas from [48], by leveraging the feasibility of building monolithic CNFET 3D ICs, 3DTUBE can support redundancy at multiple granularities tolerating failures in high-variation technologies at a minimal overhead.

4.7 Conclusion

Although breakthrough fabrication techniques to realize carbon nanotube transistors (CNFETs) have been invented, the process is still at its infancy and has a high failure rate. We require circuit and architectural reliability solutions to improve the yield and help commercialize CNFET designs, that can provide investment for fast-paced fabrication improvements. In this work, we propose the use of a 3D reconfigurable architecture, **3DTUBE**, that can provide failure protection at multiple granularities; module and pipeline-stage levels, to

improve yield and throughput of high-variation designs. We show that for a failure rate of 10 ppm, the pipeline-stage and module level solutions achieve 2× and 2.5× improvement in throughput over a CNT-based 8-core OpenSPARC T1 processor with no reliability solution, respectively, at a minimal area cost of 7.4% and frequency overhead of 8.2%. Furthermore, 3DTUBE can achieve up to 6.0× higher throughput and up to 3.1× lower EDP compared to a silicon-based design evaluated at 1 ppb transistor failure rate, which is 10,000× lower in comparison to CNFET's failure rate.

CHAPTER 5

Scheduling Techniques for Real-Time constrained Heterogeneous SoCs

The increasing use of heterogeneous chip multiprocessors in recent years has reached unprecedented levels, especially in the context of IoT and distributed edge computing. Traditional homogeneous platforms have shown fundamental limitations when it comes to enabling high-performance yet-ultra-low-power computing, regardless of the application domain of interest. For example, in the specific context of next generation of connected and autonomous vehicles, the computation capabilities that are required onboard are on par with the requirements of a "traditional" high-performance computer, but operating within much stringent power limitations. By combining the right set of hardware resources (cores, accelerators, chip interconnects and memory technology) along with an adequate software stack (operating system and programming interface), heterogeneous chips have become an effective high-performance and low-power computing alternative.

With the growing prominence of fully autonomous vehicles (ground, aerial, surface and underwater), major investments are being made into developing applications to make these vehicles efficient and safe. In order to ensure functionally correct *and* safe operation, the complexity of state-of-the-art full-stack hardware-software platforms for autonomous vehicles (AVs) have progressively increased over the last decade — specifically in the form of highly-heterogeneous hardware systems driven by highly-heterogeneous software applications. The resulting "nominal" hardware-software platform for AVs consists of

domain-specific (embedded) systems-on-chips (SoCs) with multiple acceleration engines specifically catering to ultra-efficient execution of application kernels for perception, planning, control, and communication. Examples of these platforms include NVIDIA's DRIVE AGX [51] and Tesla's Full Self-Driving Chip (FSD Chip) [52]. In this context, the existing work focuses mostly on: (i) optimal SoC platforms for AVs to comply with stipulated performance, efficiency and resiliency metrics, and (ii) the development of AV applications that can meet the increasing functionality and safety requirements for autonomy. Little attention has been paid to the aspect of process scheduling for AV applications on heterogeneous SoCs. The *de facto* approach relies on schedulers developed for:

1. Heterogeneous systems (traditional distributed systems and not SoCs, therefore low variation in execution time across processing elements) [53, 54].
2. Real-time constrained applications [55, 56].
3. Real-time constrained applications based on the ones listed in 1) above [57, 58, 59].

However, none of these schedulers use dynamic runtime information from the system to efficiently schedule real-time applications on heterogeneous SoCs. Moreover, these schedulers operate in a greedy manner, trying to meet the real-time requirements of individual processes or applications without any consideration of a *global* objective function, defined as the Quality-of-Mission (QoM) metric for AV applications. We define the QoM as a composite metric encompassing both the mission performance and the fraction of mission completed *safely*, *i.e.*, while meeting the real-time and safety constraints. To demonstrate the value of considering the QoM, we examine the following two schedulers:

1. **Quality-of-Mission-agnostic approach:** The tasks of an AV application, represented as directed-acyclic graphs (DAG), are statically scheduled using the earliest-finish-time and lower-bound approach detailed in [59]. During execution, based on the *safety-criticality level* of a DAG (a specification that indicates how critical the timely execution of a task

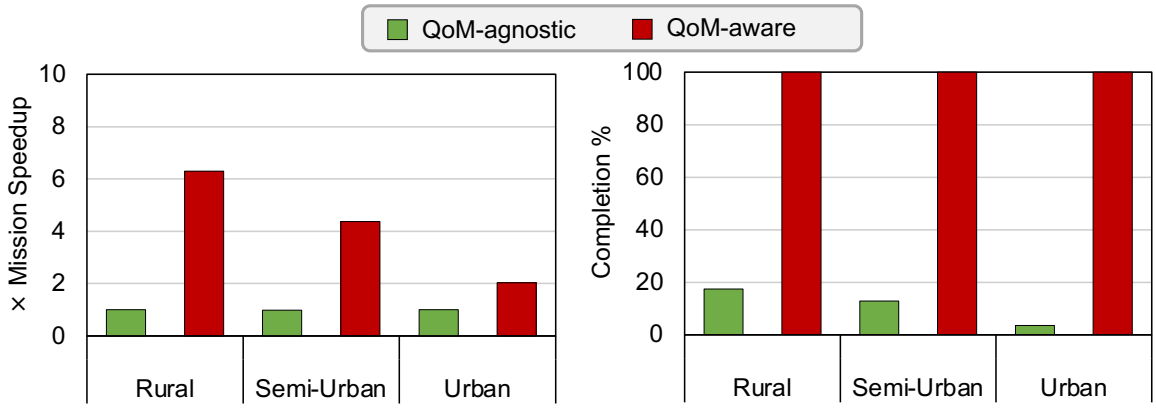


Figure 5.1: *Left-axis:* Mission speedup of Quality-of-Mission aware scheduler ("QoM-aware") over the Quality-of-Mission agnostic scheduler ("QoM-agnostic") and *Right-axis:* PE utilization of "QoM-agnostic" and "QoM-aware", when evaluated under three congestion scenarios: rural, semi-urban and urban.

is for safe AV operation), the tasks are re-ordered to execute on the fastest processing element (PE).

2. **Quality-of-Mission-aware approach:** Tasks are ranked taking into account the temporal density of safety-critical DAGs in the system, real-time constraints (deadline and criticality-level) along with their heterogeneous execution profiles and dynamic runtime information. This allows the scheduler to make "smarter" scheduling decisions across PEs in highly heterogeneous SoCs while navigating through dynamic environments, as we propose in this work.

Figure 5.1 presents the evaluation of these scheduler variants under progressively more congested navigation conditions (rural, semi-urban and urban). The figure evaluates the schedulers on two metrics: (i) the overall mission time while meeting all real-time constraints and (ii) the percentage of mission completed safely when the AV is operating at the maximum safe speed achievable among the two schedulers. The schedulers are evaluated on a simulated platform with eight general-purpose cores, two GPUs, and one fixed-function hardware accelerator. To complete a *mission* (*e.g.*, safely navigate from a starting location "A" to a destination "B"), the SoC executes a series of *applications*, composed of *tasks* (or kernels, or processes). Examples of AV tasks include perception, planning and control.

Figure 5.1 reveals that scheduling decisions can drastically affect the safe speed of the AV, and consequently its overall mission completion time across varying congestion levels. The "QoM-aware" scheduler outperforms the "QoM-agnostic" scheduler in terms of mission time by 6.3×, 4.4× and 2.0× for the rural, semi-urban and urban scenarios, respectively. Similarly, when "QoM-agnostic" is operated at the speed safely achieved by "QoM-aware", it can complete only up to 17% of the mission before leading to a hazard. This motivates the fact that a holistic approach that is aware of the heterogeneity in hardware and applications along with dynamic run-time information helps make better scheduling decisions even in a highly congested urban-like scenario. Moreover, using this information, the scheduler can stall or prune less-critical applications in favor of more critical ones, or prioritize the execution of a given task on an accelerator over other tasks which may need to use the same accelerator – the approach followed by "QoM-aware". The key observation here is that **real-time constrained execution of AV applications without accounting for hardware heterogeneity and dynamic runtime information does not necessarily translate into the best overall mission performance (*e.g.*, mission time).**

In this work, we propose a "QoM-aware" scheduler called AVSched. AVSched is a multi-level scheduler that leverages the synergy between the underlying heterogeneous hardware platform and the applications' runtime characteristics to satisfy the growing throughput demand of AVs, while meeting the specified real-time and safety constraints. Specifically, AVSched proposes two scheduling policies: MS_{static} and $MS_{dynamic}$ and scheduling optimizations like *task-traffic reduction, hierarchical hetero-ranking and rank update*. The first step in the operation of AVSched involves profiling the application tasks across *all* the possible PEs in the SoC[1]. This information is then used by AVSched to guide its scheduling decisions. AVSched also uses safety criticality information provided by the application, which is *key* to comply with safety specifications. Runtime information, gathered from hardware monitors in the SoC, are used by AVSched during its operation to

[1]Note that offline application profiling is a common approach across most of the schedulers considered in this work.

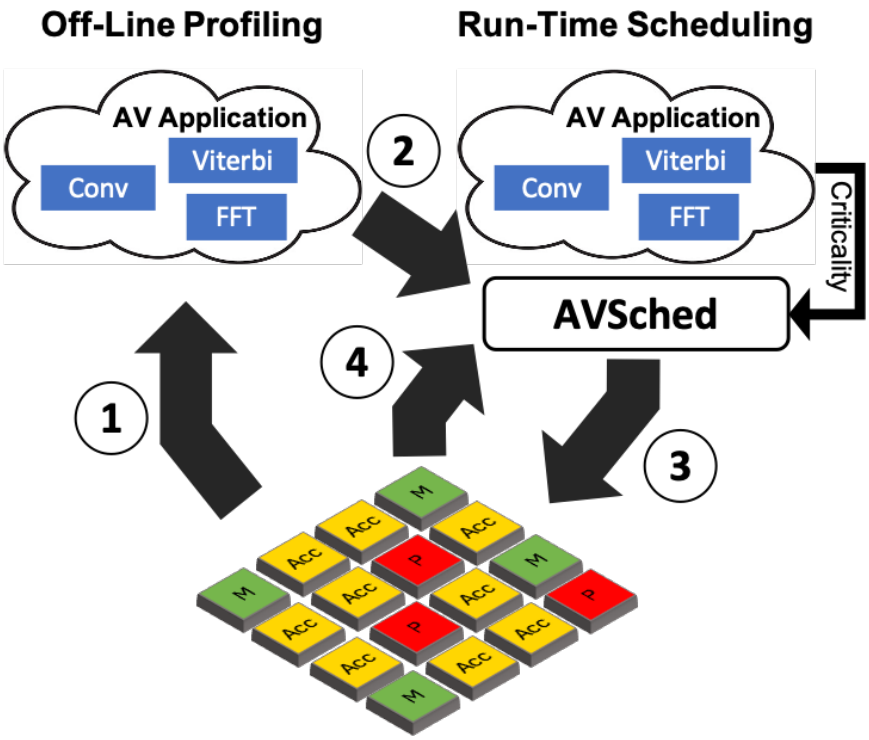


Figure 5.2: AVSched's operation workflow, including the off-line profiling and run-time scheduling stages.

keep track of real-time deadlines of the application, and to estimate data movement costs, waiting times of ready tasks, available slack for a DAG, and the power consumed by a completed task. These monitors include, but are not limited to, the status of PEs (*available* or *busy*), estimated completion time for the tasks running on busy PEs, and the execution profile of completed tasks.

Moreover, efficient design space exploration of various processing elements (PEs) in the SoC can be achieved by using AVSched to optimize for the mission time, PE utilization and energy consumption for AV applications constrained by different environmental conditions as shown in Figure 5.3.

Specifically, the novelty of this work is as follows.

1. We demonstrate that hardware heterogeneity along with the application's runtime information is key in determining the scheduler's effectiveness while unveiling new opportunities for "smarter" task scheduling.

2. We propose AVSched, a multi-level scheduler that follows a holistic approach to optimize the *Quality-of-Mission* (QoM), while meeting real-time safety constraints in autonomous vehicles. The scheduler exploits the highly-heterogeneous nature of the underlying SoC and dynamic run-time information (like maximum/minimum slack available and task wait times) to make better scheduling decisions.

3. We introduce optimizations like *task pruning*, *hierarchical hetero-ranking* and *rank update* built upon two AVSched policies (MS_{static} and $MS_{dynamic}$), that result in a performance improvement of 3.5× (average) for real-world AV applications, when compared against state-of-the-art schedulers.

4. We demonstrate significant reductions in mission time, energy consumption and SoC area, obtained with AVSched in a design space exploration (DSE) loop. We show that AVSched on average reduces the accelerator resource requirement of an efficient SoC to safely complete AV missions by 1.9× in comparison to prior state-of-the-art schedulers [58, 59].

5.1 Background and Motivation

5.1.1 Autonomous Vehicle Applications

To achieve high levels of safety, reliability and precision, AV applications are constituted of highly heterogeneous tasks that can be divided into three types based on their function: *perception*, *planning* and *control* [60]. Through perception tasks, AVs sense the environment and perform obstacle detection, localization and classification to determine further action. Planning tasks are implemented to make decisions that help achieve the vehicle's goals such as reaching a destination or searching an unknown location while ensuring safety and mission quality. Lastly, control tasks such as traction control, acceleration, braking, steering, and lane keeping are executed to follow the planned actions. We describe the characteristics of and models for AV applications in the following paragraphs.

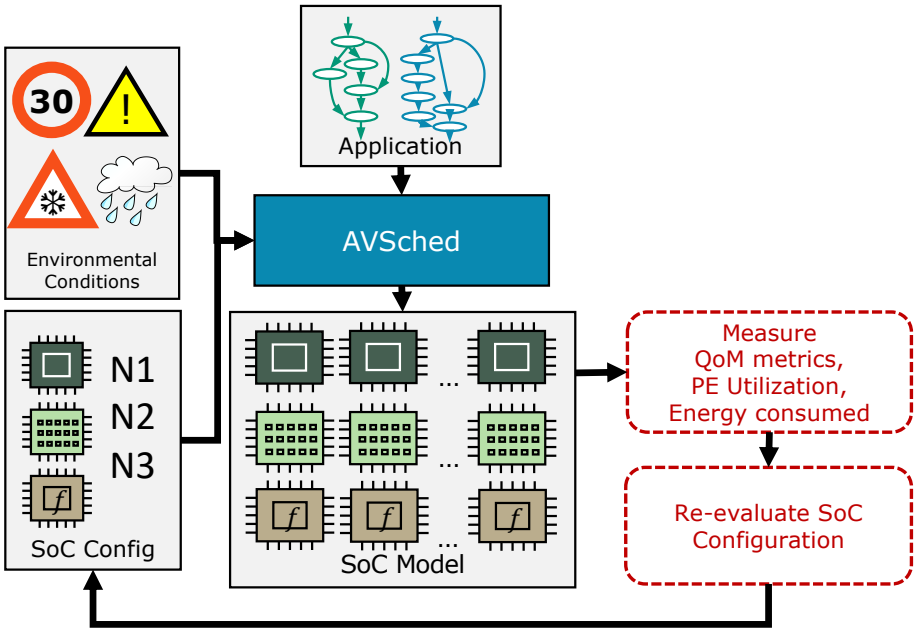


Figure 5.3: Use of AVSched to optimize SoC design for AVs while considering QoM metrics, PE utilization and energy consumption. Use of an efficient scheduler in this optimization loop helps reduce the compute requirements of an SoC while improving the mission performance

5.1.1.1 High Heterogeneity in Execution Time

Heterogeneous system-on-chip (SoC) is a single chip comprising of many processing elements (PEs). PEs are a mix of varied compute units like general-purpose processors CPUs, special-purpose processors GPUs and hardware accelerators. Heterogeneous SoC designs are extensively used in mobile and automotive industries. Increasingly they are also being adopted by domains dominated by homogeneous architectures like datacenters [61]. Heterogeneous SoCs are widely being adopted due to the heterogeneous characteristic of applications. They improve performance and power of applications while minimizing communication and data movement costs between PEs. The domain-specific SoC presents its own challenges. Although the SoC reduces data movement cost, they have limited compute resources due to power-constraints. Furthermore, task execution times can vary by a magnitude of hundreds across PEs on a heterogeneous AV SoC [62]. In our experiments, we observed task execution times can vary by up to 300× across PEs when executed on

a heterogeneous SoC (Section 5.4). Therefore, to make heterogeneity-aware scheduling decisions, we rely on an offline timing profile created for each task that can be stored on-chip along with the application binary. A task's timing profile comprises both the intra-PE *execution cost* as well as the inter-PE *data movement cost* of inputs/outputs for all eligible PEs.

5.1.1.2 Application Model

For AVs, all applications and their conforming tasks are fixed at runtime, *i.e.*, the addition of a new application (with its offline timing profile) would be provided as a software update to the AV. Based on the type of AV and its mission, these tasks are executed according to a fixed control-flow graph (CFG), where edges in the graph are dynamically decided based on the inputs and decisions made during runtime. We derive directed-acyclic graphs (DAGs) as subgraphs from these CFGs that are statically known, although the arrival and execution of these DAGs are dynamic and determined during vehicle operation. These dynamically arriving static DAGs constitute the input to the scheduler. A DAG contains nodes and edges. We map a task in a CFG to a node in the DAG and dependencies between tasks as edges. Note that a task is a unit of work that can execute independently when all its dependencies (both data and control) are resolved. The DAGs can be generated using compiler techniques for extraction of basic blocks from the CFG.

5.1.1.3 Safety Criticality Level of Applications

Depending on the environment in which the AV is operating, each iteration of the control-flow graph can execute at a different safety-criticality level. For autonomous driving applications, ISO 26262 identifies four Automotive Safety Integrity Levels (ASIL): A, B, C, and D [63]. ASIL-A represents the lowest criticality level (*i.e.*, operations which can result in no injuries), and D represents the highest criticality level (*i.e.*, operations that can result in the highest degree of automotive hazard). Similarly, unmanned aerial vehicle tasks,

have criticality levels assigned based on the Design Assurance Level (DAL) [64]. For the safe and reliable operation of AVs, it is absolutely necessary to comply with these criticality levels. In this work, we consider DAGs to belong to two criticality levels:

Non-Critical DAGs: those with criticality 1 (`crit=1`) that arrive periodically to the system. They are equivalent to ASIL A, *e.g.*, object recognition on a blind-spot camera while traveling straight on a single-lane road.

Critical DAGs: those with criticality 2 (`crit=2`), that are classified as critical in two ways:

- Promoted DAGs: If no `crit=1` DAG meets its deadline within a time period T_{crit}, then the scheduler can promote it to `crit=2` in order to provide redundancy and avoid potential hazards in the AV operation, *e.g.*, a path-planning operation that uses GPS to calibrate the location of the AV while it is moving along a straight line.

- Highly-critical DAGs that represent applications of ASIL levels which can result in a clear safety hazard (B, C, and D) would be `crit=2` DAGs, *e.g.*, forward-camera perception of a stop sign during forward motion.

For safe AV operation, DAGs with `crit=2` must be executed within specific *hard deadlines* to avoid potential hazards. DAGs with `crit=1` have *firm deadlines*, *i.e.*, if executed within their deadlines they could help improve the mission. Otherwise, the output of the DAG is not useful.

5.1.2 Congestion in Environmental Conditions

The safety and resilience of AVs are of significant importance, due to the high toll they can have on human lives and infrastructure [65, 66]. Hence, the assessment of AV systems operating in varying dynamic scenarios is absolutely necessary [67, 68, 69]. The congestion of an environment is determined by the temporal density of `crit=2` DAGs encountered during execution. This can be influenced by conditions like the weather, traffic and terrain. For example, in the case of autonomous driving, the vehicle might encounter several

crosswalks while driving from point A to point B in an urban area. In this case, the AV passing each crosswalk could be accompanied by the arrival of a `crit=2` DAG. Therefore, the more congested the environment (*e.g.*, more crosswalks), the higher the number of `crit=2` DAGs that the AV will have to execute. Based on the traffic experienced by an AV, we determine three congestion scenarios; *rural*, *semi-urban* and *urban*, however, these can be extended to congestion scenarios caused by any other environmental conditions.

5.1.3 Application Deadline and Speed of the AV

The speed of the AV determines the rate at which DAGs arrive at the scheduler. Each DAG is also associated with a *real-time deadline*, determined by the speed of the vehicle (the faster the AV travels, the tighter the deadline), congestion in the environment and criticality of the application. Hence, the speed of the AV is directly proportional to the rate at which DAGs arrive, and inversely proportional to the deadline. The maximum arrival rate at which the AV meets 100% critical deadlines is considered equivalent to the *maximum safe speed* at which the AV can operate in each congestion scenario.

5.1.4 Quality-of-Mission (QoM) Metrics

Various figures of merit can be used to measure an AV's mission quality. In this work, we use universal metrics that can be applied to all autonomous vehicles, similar to the ones adopted in [70]. We choose the following QoM metrics to evaluate our scheduler for varying congestion scenarios:

- *Mission time* to complete the objective of the mission, *e.g.*, navigation time from location A to location B, while complying with safety requirements of meeting deadlines for all DAGs with `crit=2`.
- *Fraction (or %) of mission completed* at a given speed before missing the first `crit=2` DAG deadline. For example, if the best scheduler can complete a mission safely while

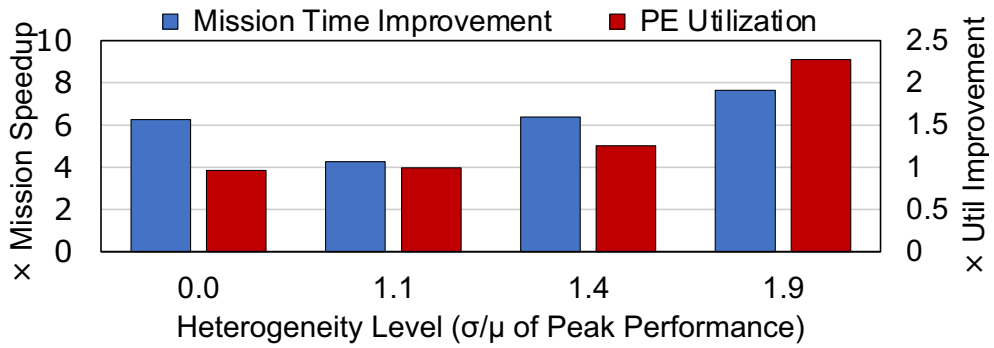


Figure 5.4: Effect of SoC heterogeneity, calculated as the coefficient of variation in PE peak performance, on mission speedup and PE utilization of a Quality-of-Mission-aware scheduler ("QoM-aware") over a Quality-of-Mission agnostic ("QoM-agnostic"). We increase heterogeneity in the system by considering varying PE configurations in the SoC.

operating at a maximum speed of S, then this metric for the scheduler being evaluated is calculated as the % of total critical DAGs of the mission an AV running at speed S can complete before it fails to meet the deadline of a critical task leading to a hazard.

5.1.5 Domain-Specific Systems-on-Chips

High heterogeneity in AV applications, real-time constraints, and the demand to process multiple critical applications call for the use of highly heterogeneous systems. These SoC platforms consist of multiple processing elements (PEs) with different performance and efficiency characteristics; namely, CPUs, GPUs, accelerators, *etc.* [52, 71, 72, 73]. Heterogeneous SoCs accelerate the execution of a task by providing increased computational capabilities, reduced data movement cost between PEs, and reduced need to offload computation to cloud servers (or other vehicles, in case of connected vehicle systems [74, 75]), in addition to higher energy efficiency.

The heterogeneity in PEs (Table 5.4) results in new challenges and opportunities when allocating on-chip resources or making task scheduling decisions. To illustrate the need for global schedulers that are aware of the heterogeneity in an AV SoC, we compare the quality-of-mission agnostic scheduler ("QoM-agnostic") with the quality-of-mission aware scheduler ("QoM-aware"), as described in Section 5, for different hardware configurations.

Table 5.1: Real-Time and Heterogeneous Schedulers.

Prior Art	Input	Ranking Type, Parameters	Hetero-Aware	QoM-Aware
CPATH [54]	Single Dynamic DAG	Dynamic Critical path length	×	×
RHEFT [57]	Multiple Static DAGs	Static, Earliest finish time, relative laxity degree	×	×
2lvl-EDF [58]	Multiple Static DAGs	Dynamic Earliest deadline first	×	×
ADS [59]	Multiple Static DAGs	Static, Earliest finish time, criticality	×	×
AVSched	Multiple Static DAGs	Dynamic, Deadline, PE variation, criticality	✓	✓

We use the *coefficient of variation* [76] of the PE's peak performance as a proxy for the heterogeneity level in the SoC. Figure 5.4 shows that as we increase heterogeneity (by diversifying the PEs), "QoM-aware" is able to improve performance by up to 7.6× over "QoM-agnostic". By leveraging this heterogeneity information, "QoM-aware" is also able to improve PE utilization by up to 2.2× over "QoM-agnostic". The takeaway is that **synergistic exploitation of the underlying hardware, the application's real-time requirements (deadline and criticality) and dynamic runtime information can significantly improve mission time and hardware utilization**.

5.1.6 State-of-the-Art Real-Time *and* Heterogeneous Schedulers

The processing time of any task comprises of four components: the transfer of input data to the PE that will execute the task (*data movement time*), the time required to make the scheduling decision (*scheduling decision time*), the time spent while the task is waiting to be executed on the scheduled PE (*waiting time*), and the time to execute the task on the scheduled PE (*execution time*). In order to minimize the mission time of an AV, it is critical to reduce all four components. While data movement and execution time are significantly reduced with the use of heterogeneous SoCs, all four components are also highly dependent on the scheduling algorithm. Prior schedulers developed for heterogeneous

data-center architectures do help curtail the processing time, but are neither *hetero-aware* (do not efficiently schedule tasks with high-variation in timing profile on an SoC) nor do they optimize for stringent real-time and safety constraints. Furthermore, schedulers developed for real-time-constrained applications are not flexible enough to provide the best QoM metrics or efficiently utilize the underlying hardware. Table 5.1 provides a comparison of this work with prior art and briefly discusses them here.

CPATH [54] is a scheduler that prioritizes tasks in the DAG based on a bottom-cost longest-path approach and submits high priority tasks to fast cores and low priority tasks to slow cores with work-stealing enabled. CPATH aims to optimize the response time of a single DAG. When applied to a multi-DAG application with real-time constraints, it fails to meet deadlines at higher arrival rates of DAGs. In contrast, our work targets to meet deadlines in safety-critical multi-DAG scenarios.

RHEFT [57] schedules multiple DAGs by calculating the latest start time and sub-deadline of each task and pre-scheduling all DAGs using a rank based on HEFT [53] and the relative laxity degree. However, RHEFT does not consider safety constraints leading to higher mission time. In contrast, AVSched employs efficient non-critical DAG pruning boosted by hierarchical heterogeneous ranking to improve mission time.

2lvl-EDF schedules tasks with the earliest deadline on the earliest finish time PE, as described in [58]. Similar to RHEFT, it neither considers safety constraints of tasks nor the variation in the timing profile of tasks on the heterogeneous SoC with respect to deadlines.

ADS schedules ranked DAGs based on [53] and dynamically prioritizes tasks with higher criticality levels, as described in [59]. However, ADS neither predicts when to prune non-critical tasks, nor is hetero-aware. AVSched is able to outperform this policy by pruning non-critical tasks, which is enabled by AVSched's hierarchical heterogeneous ranking optimization.

None of these prior schedulers operate efficiently on highly-heterogeneous SoCs while optimizing for the real-time requirements of the application and improving overall AV

mission performance. Our work targets to fill this void.

5.2 AVSched: Mission & Heterogeneity-Aware Scheduler

AVSched is a multi-level scheduler that exploits the heterogeneous nature of domain-specific SoCs to improve QoM and PE utilization for AV applications. Specifically, AVSched consists of two levels: `Meta-Sched` and `Task-Sched`. `Meta-Sched` translates the mission and application (DAG) level information to tasks, while `Task-Sched` performs the actual task-to-PE assignment and resource management. The two layers communicate using a set of data structures: a *ready queue*, a *completed queue* and a *prune list*.

As depicted in Figure 5.5, when DAGs arrive for execution, `Meta-Sched` tracks task dependencies, prioritizes ready tasks based on a rank, and performs non-critical task pruning. `Task-Sched` receives ready tasks from `Meta-Sched` and updates the ranks of the tasks, populates the prune list, assigns tasks to PEs, and sends information about completed tasks back to `Meta-Sched`.

5.2.1 Meta-Sched and Task-Sched Operations

This section describes `Meta-Sched` and `Task-Sched` operation and introduces the various scheduling features in AVSched. Some key terms are defined in Table 5.2.

Table 5.2: Description of parameters used in AVSched.

Abbreviation	**Parameter Description**
WCET / BCET/ ACET	Task's worst/best/avg.-case execution time across all PEs
EET	Task's estimated execution time
PT	Sum of all WCET of tasks in the path
CPT	Sum of all WCET of tasks in the critical path
CPST	Sum of all WCET of tasks in the segment of the path that intersects the critical path
NCPST	Sum of all WCET of tasks in the segment of the path that does not intersect the critical path
SD / SDR / SR	Task's sub-deadline / sub-deadline ratio / slack ratio

5.2.1.1 Dependency Tracking

`Meta-Sched` processes DAGs to find *ready* tasks. A task is determined as *ready* when all its parent nodes in the DAG have completed execution, *i.e.*, it has no incomplete dependencies (incoming edges) in the DAG. Therefore, when a task completes execution on a PE, `Meta-Sched` resolves edges to the child tasks and marks those with no remaining dependencies as ready.

5.2.1.2 Task Prioritization

Each application (DAG) arriving at `Meta-Sched` has an associated deadline and criticality. Moreover, these DAGs have varying structures in terms of the number of tasks, types of tasks (execution profile) and dependencies (edges). Therefore, to make an informed task scheduling decision (*i.e.*, considering the real-time constraints of the DAG and mission performance), `Meta-Sched` assists `Task-Sched` by assigning ranks to ready tasks and ordering them. The rank encodes DAG- and mission-level information as it is determined using the deadline of the parent DAG (the DAG to which the task belongs), criticality and structure, and the task's execution profile. A task's *rank* is calculated as:

$$Rank = \frac{Criticality}{Slack}; \quad Slack = SD - EET \tag{5.1}$$

where, $Criticality$ is the criticality of the task determined by that of the parent DAG, and $Slack$ is the task's effective slack calculated by `Meta-Sched` as the task's sub-deadline (SD) minus its estimated execution time (EET). Therefore, tasks with higher criticality and smaller slack to their deadline are given higher priority. We explore multiple rank assignment policies based on the way SD and EET are computed. We use the parent DAG's structure and the task's execution profile to determine SD, *i.e.*, for the path within the parent DAG on which the task lies, we find the worst-case execution time ($WCET$) of the path and of the task. The $WCET$ of a task is the time required execute the task

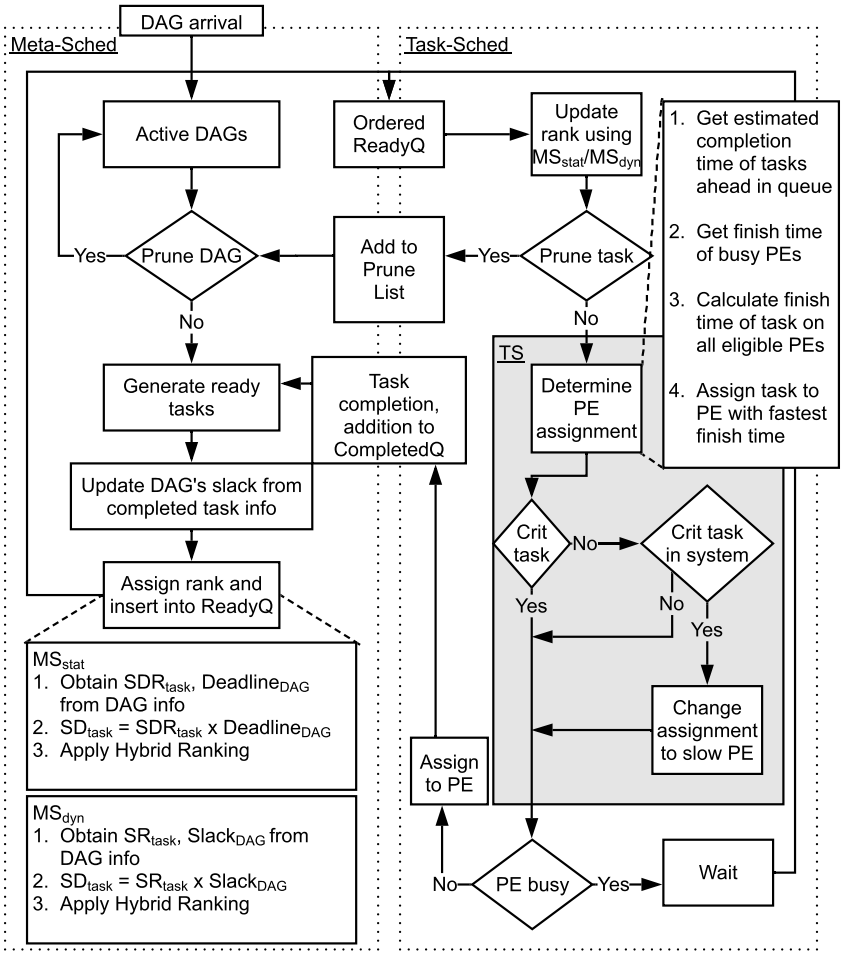


Figure 5.5: AVSched operations showing `Meta-Sched` (mission & DAG processor) and `Task-Sched` (task scheduler & hardware manager), and their synchronization using the ready and completed queues, and prune list.

on the slowest PE. Therefore, by using $WCET$ to calculate SD, AVSched allows for the tasks to be scheduled on any available PE in the system, whereas using average or best-case execution time ($BCET$) can bias the scheduler's decision towards faster PEs. Depending on the way SD is calculated, AVSched policies are classified into MS_{static} and $MS_{dynamic}$.

MS_{static}: In the MS_{static} policy, we determine SD as a weighted ratio of the DAG's deadline ($Deadline_{DAG}$). This weighted ratio, called the task's sub-deadline ratio (SDR), is calculated as the task's $WCET$ relative to its path's execution time. Since each DAG and timing profile of the tasks in it are statically known, SDRs can be calculated offline and

stored along with the timing profile of the DAG. For a given DAG, the path time (PT) is calculated as the sum of the $WCET$s of the tasks in that path. The critical path time (CPT) is the one with the largest PT. A task's SDR and SD calculation are based on the path of the DAG on which it lies:

- If the task lies on the critical path or on a path that does not intersect with the critical path, SDR is calculated as the ratio of the $WCET$ of the task to the path time PT:

$$SDR = \frac{WCET}{PT}; \;\; SD = SDR \times Deadline_{DAG} \tag{5.2}$$

- If the task lies on a path which intersects with the critical path, then the path is divided into two segments, the critical path segment (taking a critical-path segment time, $CPST$) and the non-critical path segment (with non-critical-path segment time, $NCPST$). For tasks on the $NCPS$, we first calculate the deadline allocated to the $NCPS$, $Deadline_{NCPS}$, as the $Deadline_{DAG}$ minus $Deadline_{CPS}$. Using Equation 5.2,

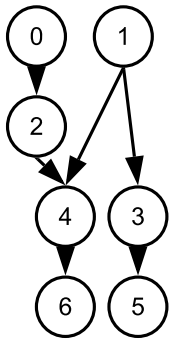
$$Deadline_{CPS} = \frac{CPST}{CPT} \times Deadline_{DAG} \tag{5.3}$$

$$Deadline_{NCPS} = Deadline_{DAG} - Deadline_{CPS} \tag{5.4}$$

  For tasks on the $NCPS$, SDR and SD are calculated similarly, using a path time of $NCPST$ and deadline of $Deadline_{NCPS}$ as:

$$SDR = \frac{WCET}{NCPST}; \;\; SD = SDR \times Deadline_{NCPS} \tag{5.5}$$

If a given task's sub-deadline SD can be calculated using several of the above methods, we pessimistically assign it the smallest of the computed SD values. To illustrate with an example, Figure 5.6 shows a small, 7-task DAG. Let path P_0, the path consisting of tasks 0,



Task	0	1	2	3	4	5	6
CPU	500	200	300	100	400	100	200
GPU	100	40	100	-	100	50	50
Accel	20	10	-	10	50	5	-

Figure 5.6: *Left:* A 7-task DAG containing three paths: P_0, P_1 and P_2. P_0 contains tasks 0, 2, 4 and 6; P_1 contains tasks 1, 4 and 6; and P_2 contains tasks 1, 3 and 5. ***Right:*** Timing profile for each task on three types of PEs: CPU, GPU and an accelerator. Using the timing profile, we determine that P_0 is the critical path, P_1 intersects the critical path, and P_2 is independent of it.

2, 4 and 6 be the critical path. P_1 contains tasks 1, 4 and 6 and P_2 is composed of tasks 1, 3 and 5. While P_1 intersects the critical path, P_2 does not. Therefore, SDs for tasks on P_0 and P_2, are calculated using Equation 5.2. Since the SD for task 1 on the NCPS of P_1, SD can also be calculated using Equation 5.5, we assign it the lower value of the two.

$MS_{dynamic}$**:** In this policy, we assign the task's sub-deadline based on a dynamic metric of the DAG. Specifically, SD is calculated using the DAG's available slack ($Slack_{DAG}$): the deadline *remaining* for a DAG during execution, when that ready task's rank is being calculated, as opposed to MS_{static} that uses a static distribution of the DAG's deadline to calculate the task's SD. Therefore, $MS_{dynamic}$ accounts for tasks in the DAG that might have exceeded their sub-deadlines. $MS_{dynamic}$ adjusts the SD of ready tasks based on the available slack of the DAG by calculating a task's $WCET$ relative to the execution time of tasks *remaining* in the task's path:

$$SR = \frac{WCET}{\sum WCET_i}; \quad SD = SR \times Slack_{DAG}. \tag{5.6}$$

where, SR is the task's slack ratio and $WCET_i$ is the $WCET$ of each remaining task i that lies on the same path as the task, including the task itself. If a task lies on multiple paths, then lowest SR calculated across all paths is selected. For the DAG in Figure 5.6, we

Algorithm 1: Heterogeneous Ranking

```
1  foreach task in ReadyQ do
2      max_slack = SD − BCET
3      min_slack = SD − WCET
4      if min_slack >= 0 then
5          eff_slack = min_slack
6      else if max_slack >= 0 then
7          eff_slack = max_slack
8      else
9          eff_slack = 1/max_slack
10     end
11 end
12 task.rank = criticality/eff_slack
13 sort readyQ based on task.rank
```

calculate SD and SR of the tasks using Equation 5.6. Since task 1 lies on two paths, P_1 and P_2, its SR is calculated twice, and we choose it to be smaller of the two.

Heterogeneous Ranking: Most prior schedulers either use the $WCET$ or the average-case execution time ($ACET$) to compute EET. However, due to the large variation in execution time of a task across different PEs, we instead introduce a heterogeneous rank (abbreviated as *hetero-rank*) that dynamically chooses EET to be either $WCET$ or $BCET$, based on the deadline and execution time of the task. This allows us to prioritize tasks closer to their deadlines based on the underlying hardware characteristics. It also allows to prioritize tasks that have just missed their deadline (smaller negative slack) over a task that missed its deadline long ago (large negative slack). The precise algorithm is presented in Algorithm 1.

Hierarchical Ranking: In order to differentiate between tasks with different criticalities and execution patterns across the different hetero-rankings, we introduce a hierarchical ranking scheme (denoted as $rank\_type$) and describe it in Algorithm 2. Note that for a critical task that has negative slack, we set its $rank\_type$ to the maximum (5). For a non-critical task in the same scenario, we set its $rank\_type$ to the minimum (0). This allows AVSched to prioritize critical tasks on fast PEs and execute non-critical ones on slower PEs, with the goal of improving overall mission performance while respecting safety constraints. Similarly, critical tasks with smaller slacks are prioritized over critical tasks with larger slacks and vice-versa for non-critical tasks, so that fast PEs are not expended to meet non-critical task

Algorithm 2: Hierarchical Heterogeneous Ranking

```
1  foreach task in ReadyQ do
2      max_slack = SD − BCET
3      min_slack = SD − WCET
4      if task.crit = 2 then
5          if min_slack >= 0 then
6              eff_slack = min_slack; task.rank_type = 3
7          else if max_slack >= 0 then
8              eff_slack = max_slack; task.rank_type = 4
9          else
10             eff_slack = 1/max_slack; task.rank_type = 5
11         end
12     else
13         if min_slack >= 0 then
14             eff_slack = min_slack; task.rank_type = 2
15         else if max_slack >= 0 then
16             eff_slack = max_slack; task.rank_type = 1
17         else
18             eff_slack = 1/max_slack; task.rank_type = 0
19         end
20     end
21 end
22 task.rank = criticality/eff_slack
23 sort readyQ based on task.rank_type and task.rank
```

deadlines in the presence of critical tasks. Based on these rank assignments, `Meta-Sched` orders ready tasks into the ready queue.

5.2.1.3 Rank Update

`Task-Sched` receives ordered ready tasks from `Meta-Sched`. Before the task assignment, the ranks of the tasks waiting in the ready queue are updated to subtract the time elapsed since previous update from the current effective slack ($Slack$). The tasks are then re-ordered according to the updated ranks. Updating the ready tasks' ranks can also help in finding non-critical tasks that might not meet their deadline, which can then be considered candidates for pruning, reducing the overall system task traffic. `Task-Sched` passes these tasks to `Meta-Sched` for potentially pruning their parent DAGs using the *prune list*.

5.2.1.4 Task Assignment

`Task-Sched` uses a task assignment policy to assign ordered ready tasks in the ready queue to eligible PEs (*i.e.*, PEs that can execute the task). Furthermore, once a task completes execution, it is pushed into the completed queue along with information about the PE on which it was executed and the timestamp at which it completed execution. In this work, we introduce a non-blocking task assignment policy, called **TS**,

that schedules a task in the ready queue to the PE that will result in the earliest estimated finish time for the task, factoring in the execution time of the task, current busy status of the PE and all tasks ahead of this task in the ready queue that are potentially scheduled to the same PE as shown in Figure 5.5. `TS` chooses the task to be scheduled using a non-blocking task assignment policy within a window of size w, thus searching w tasks past the head of the queue that could potentially be waiting for the earliest estimated finish time PE to become available. `TS` is also aware of the timing profile and criticality of each task. Therefore, if the task is non-critical and critical tasks are present in the system, `TS` can effectively improve utilization by scheduling this task on available slow PEs (Figure 5.5).

5.2.1.5 Completed Task Information

Once a task completes execution, `Task-Sched` pushes it into the completed queue along with information about the PE on which it was executed and the timestamp at which it completed execution. This information is used by `Meta-Sched` for dependency tracking and to obtain the data movement cost of children tasks.

5.2.1.6 Deadline Tracking and Task Pruning

`Meta-Sched` can elect to prune DAGs, *i.e.*, not execute them at all/any further, thus eliminating non-critical tasks that will not meet their deadlines in order to reduce traffic in the system. After the execution of each task, when `Meta-Sched` searches the existing DAGs for ready tasks, it also calculates the estimated slack available for each DAG, assuming

that ready tasks can execute at their best-case execution time ($BCET$). If the estimated slack available is negative and the DAG has `crit=1`, the entire DAG is pruned. `Meta-Sched` also prunes DAGs based on the tasks in the prune list, identified during the rank update process. Note that in Algorithm 2, `Meta-Sched` prunes DAGs that have $rank\_type$ 0 or 1, *only if* there are critical DAGs in the system.

5.2.2 Summary of Cumulative AVSched Features

In summary, AVSched introduces the following scheduling policies and optimizations.

5.2.2.1 Task-Sched policy, `TS`

A non-blocking scheduler, that schedules ready tasks to the PE with the fastest projected finish time. `TS` also schedules non-critical tasks to the slowest PEs, if critical tasks are present in the system.

5.2.2.2 Two-level scheduling policies

With pruning of non-critical tasks estimated to miss their deadlines. These scheduling policies prioritize ready tasks based on their rank, calculated using the criticality, sub-deadline and estimated execution time of the task, and use `TS` for their `Task-Sched`.

MS_{static} determines the sub-deadline of a task statically from the parent DAG's deadline. MS_{static} performs best when the deadlines of DAGs are significantly large, *i.e.*, when DAGs complete execution with large remaining slack and all tasks complete within their assigned sub-deadlines.

$MS_{dynamic}$ uses the task's parent DAG's available slack, computed during execution, to dynamically calculate the sub-deadline of the task. Due to this ability to adapt to changes in execution time of tasks, including missing task deadlines, $MS_{dynamic}$ performs best for stringent DAG deadlines.

5.2.2.3 Scheduling optimizations for MS_{static} and $MS_{dynamic}$

Heterogeneous ranking accounts for the variation in execution time of a task on the heterogeneous SoC using dynamic calculation of the rank via Algorithm 1.

Hierarchical ranking uses the criticality of the tasks along with hetero-ranking to improve overall mission performance by incorporating the state of the system.

Rank update revises the task ranks to incorporate time waiting in the ready queue or when critical tasks are encountered. This feature also identifies non-critical tasks that will not meet their deadlines and should be pruned.

5.3 Experimental Methodology

5.3.1 Hardware Description

To evaluate AVSched, we first profile (offline) a set of AV kernels on an NVIDIA Jetson TX1 board, which is representative of an SoC used in real-world AV systems. This information is then used to simulate a heterogeneous SoC with multiple PEs. We assume that the simulated SoC has variants of the Arm Cortex-A57 CPU and the NVIDIA Maxwell GPUs with 256 CUDA cores, and fixed-function accelerators for certain tasks. We consider three systems, named Sys_A, Sys_B and Sys_C, the hardware descriptions of which are shown in Table 5.3. Note that Sys_C is composed of hypothetical PEs and is only used to demonstrate the benefit of AVSched with highly-heterogeneous systems. We consider a unified memory (shared physical address space) between the PEs in the simulated SoC, since we profiled the applications the TX1 that has unified memory between the CPUs and GPUs; AVSched, however, is not limited to this specific choice.

Table 5.3: System Description of SoC for Workload Evaluation.

Workload	System	Description
Synthetic	Sys_A	8 single-core Arm Cortex-A57 CPUs 2 NVIDIA Maxwell GPUs 1 CNN/FFT accelerator [77, 78]
Real-World	Sys_B	8 single-core Arm Cortex-A57 CPUs 2 NVIDIA Maxwell GPU 1 tracking accelerator [62] 1 localization accelerator [62] 1 detection accelerator [62]
	Sys_C	4 single-core Arm Cortex-A57 CPUs 4 single-core 25% faster CPUs (hypothetical) 1 NVIDIA Maxwell GPU 1 2× faster GPU (hypothetical) 1 tracking accelerator [62] 1 localization accelerator [62] 1 detection accelerator [62]

5.3.2 Application Task Profile

5.3.2.1 Synthetic Application Tasks

Our synthetic applications are comprised of three types of tasks: 2D Fast Fourier Transform (fft), 2D convolution (conv) and Viterbi decoding (decoder), taken from the Mini-ERA benchmark suite [79], which simulates a simplified AV with minimal environmental conditions. These tasks are representative of common AV applications, such as radar/LIDAR processing, vision/image processing and radio communication kernels. For fft, we use the FFTW3 [80] implementation for the CPU and cuFFT for the GPU. For conv, we use the Arm Compute Library implementation [81] powered by Neon SIMD extensions for the CPU, and cuDNN 5.1 for the GPU. We also obtain timing profiles of fft and conv for the accelerator designs from [77, 78]. Finally, for decoder, we use the GNURadio Viterbi function for the CPU [79] and a PyCUDA accelerated implementation [82] on the GPU.

5.3.2.2 Real-World Application Tasks

We consider two real-world AV benchmarks; ADSuite and MAVBench.

ADSuite [62] provides an autonomous driving application comprised of kernels like object detection (DET), object tracking (TRA), localization (LOC), mission planning and motion planning. For DET, we use YOLOv3 [83], a DNN-based detection algorithm, on a series of 7 images derived from the VOC dataset [84]. We use the Tiny-YOLOv3 pre-trained set of weights, which is much faster and lightweight, but less accurate compared to the regular YOLO model. For TRA, we use GOTURN [85], a DNN-based single object tracking algorithm, on a series of 14 videos in the ALOV++ dataset [86]. For LOC, we use ORB-SLAM [87], a highly-ranked vehicle localization algorithm, on 3 sequences from the KITTI datasets [88]. Further, for our GPU evaluation, we adopt the ORB-SLAM implementation in [89], where the hot paths are rewritten using CUDA. We also obtain timing profile of DET, TRA and LOC on their respective accelerators from [62]. For motion and mission planning, we use the `op_local_planner` and `op_global_planner` [90] kernels in Autoware [91]. The fusion kernel combines the coordinates of the objects being tracked with the AV location. It has a small latency, for which we only consider CPU execution.

MAVBench [70] provides a set of computational kernels that form the building blocks of many aerial vehicle applications. For some of the kernels, we use different algorithms than the ones in [70], to better exploit the heterogeneity in our hardware. Specifically, for the perception, tracking and localization kernels, we reuse our ADSuite implementations. For occupancy map generation, we use OctoMap [92] and GPU-Voxels [93] for the CPU and GPU implementations, respectively. OctoMap performs 3D occupancy grid mapping, and GPU-Voxels is a CUDA-based library for robotics planning and monitoring tasks. We generate a map composed of $200\times200\times200$ voxels. Point cloud generation and collision check consume $O(10)$-$O(1000)\times$ lower latency in comparison to other kernels (Table I in [70]), and so we only employ CPU implementations for these. For the shortest-path planners, we use the CPU-based parallel RRT (pRRT) [94] implementation in the Open Motion Planning Library (OMPL) [95], on the "Cubicle" benchmark. For the GPU implementation, we use a Poisson-disk samples-based GPU algorithm [96]. For frontier

exploration, we use the RRT ROS package that implements a multi-robot RRT-based map exploration algorithm [97], on the "single_simulated_house" scenario. Finally, we consider only CPU execution for path tracking as it has a low latency.

We consider two applications from MAVBench in this work, namely Package Delivery, where an aerial AV navigates through an obstacle-filled environment to reach its destination, deliver a package and return to its origin, and 3D Mapping, that instructs the aerial AV to build a 3D map of an unknown polygonal environment specified by its boundaries.

5.3.2.3 Data Movement Cost

In order to build a realistic input model to evaluate AVSched, we profiled the data movement time across each pair of PE types in the system. We consider the cost for data movement *within* a PE to be zero, *i.e.*, if two dependent tasks execute on the same PE, there is no additional data movement overhead. The data movement cost is characterized for CPU cores that have private L1 caches. Data movement between a CPU core and a GPU is assumed to be equivalent to the time to flush the parent tasks' output from the CPU's caches into main memory, thereby allowing the GPU to directly load the input data of the child task from the same memory location, since the CPU and GPU share the same physical address space. The data movement cost from a GPU to a different PE is encapsulated in the timing profile of the task on the GPU. For an accelerator, we consider the data movement cost from/to the accelerator to be the direct-memory access (DMA) transfer cost since many accelerator designs have their own local memory. We derived empirical data movement cost for the CPU and GPU by profiling the TX1 board, and use DMA transfer rates from published specifications [98] for the accelerators.

In profiling the data movement time, we consider the cost for data movement within a PE to be zero — *i.e.*, if two dependent tasks execute on the same PE, there is no additional data movement overhead. The data movement cost between CPU cores is assumed equivalent to flushing the parent task's *dirty* data from the private L1 caches. Data movement between

a CPU core and a GPU would be the time equivalent to moving the parent task's output from the CPU's caches into main memory, allowing the GPU to directly load from the same location for the input data of the child task. We pessimistically evaluate these to be the time to flush the CPU's data caches. Furthermore, since many accelerator designs have their own local memory, we consider the data movement cost from/to an accelerator to have an additional direct-memory access (DMA) transfer cost. Thus, if a child task C, scheduled to run on accelerator, has parent tasks A and B running on a CPU core and on an accelerator, respectively, the net data movement cost is:

$$\text{time}_{cf,A} + \text{time}_{\text{DMA},A} + 2 \cdot \text{time}_{\text{DMA},B} \tag{5.7}$$

where $\text{time}_{cf,A}$ is the cache-flush time for A from CPU to main memory, $\text{time}_{\text{DMA},A}$ is the time to DMA A's output data from main memory to the accelerator, and $\text{time}_{\text{DMA},B}$ is the time to DMA B's output data from the GPU to main memory (assumed equal to the time to DMA from main memory to the accelerator). We derived empirical data for the CPU and GPU by profiling the TX1 board and use data from the AXI specification [98] to compute the communication cost for accelerators using a simple model.

5.3.2.4 Resource Contention

The execution of a task on a given PE in a realistic SoC is naturally impacted by the volume of parallel tasks executing across the PEs, *i.e.*, the contention that the given PE faces due to resources shared with other PEs in the system, such as conflicts at the interconnect network, reduced effective cache capacity if there is no data sharing, *etc.* We used gem5 [99] to simulate a system with $N + 1$ PEs, in order to model the contention due to N PEs. The cache hierarchy and memory are modeled after the TX1 SoC. We constructed an analytical model of the contention cost across the problem sizes used in the evaluated applications, which we used while simulating the benchmarks on AVSched.

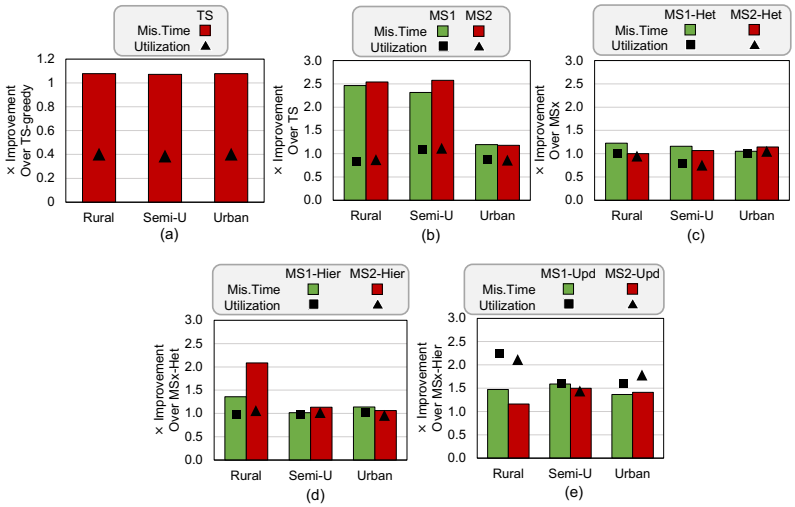


Figure 5.7: Incremental improvement in mission time and PE utilization on Sys_A of (a) 1-level task scheduler `TS` over `TS-greedy` (b) 2-level task schedulers with ranking (`MSx`) over `TS`, (c) hetero-ranking enabled `MSx` over `MSx`, (d) hier/hetero-ranking enabled `MSx` over hetero-ranked `MSx`, (e) update, hier/hetero-ranking enabled `MSx` over hier/hetero-ranked `MSx`. Note that MS1 is MS_{static} and MS2 is $MS_{dynamic}$

5.3.3 Energy and Power Model

Power/Energy Estimations. We profiled the average power consumption of each task by measuring the power consumed on the VDD rails of the CPU and GPU of the TX1 board. For the accelerators, we use the estimated power values reported in the prior work (Section 5.3.1). To compute the end-to-end energy of the SoC with AVSched, we sum the energy consumed by each task on a PE.

Dynamic Voltage-Frequency Scaling (DVFS). We apply DVFS techniques, similar to those in [100, 101], on the PEs to recuperate low utilization using a fraction of the available slack considering each task's sub-deadline, which is dependent on the scheduling policy – MS_{static} or $MS_{dynamic}$ (Section 5.2.1). Prior to a task being scheduled onto a PE, the target clock frequency is selected based on: (i) the estimated slack, (ii) the current clock frequency, and (iii) a factor f_{slack} that defines the fraction of slack to be recuperated. Note that naïvely applying DVFS on the full estimated slack (f_{slack}=1) *may* lead to deadline violations and consequently failure of the mission — *e.g.*, if a task T_i, running on a GPU, is slowed down

too much, a critical task T_{i+1}, that was formerly waiting on this GPU, could be scheduled onto a slower core. For the purposes of this work, we pessimistically apply a static value for f_{slack} across all DAGs and tasks in a DAG. In the real world, DVFS governors can be integrated within the scheduler to dynamically select f_{slack} per task based on the current PE utilization.

We enable DVFS only for the CPU and GPUs, since we observed that DVFS for even small values of f_{slack} on accelerators leads to low energy savings and mission failures. We use voltage (V_{DD}) and clock frequency (f) points from the embedded DVFS tables in the TX1 to obtain scaling factors for the PE voltage and clock. We further assume in our evaluation that the execution time of a task scales linearly with the PE's operating frequency. Finally, we assume the power to scale as $V_{DD}^{2.5}$ for estimations of energy savings using DVFS [102].

5.3.4 Trace generation

5.3.4.1 Synthetic DAG Traces

To evaluate the generality of AVSched, we generate synthetic traces of DAGs arriving at the scheduler that represent applications executed by AVs for varying congestion scenarios. Each entry of the trace consists of the arrival time, type, criticality and deadline of the DAG.

The type of DAG is determined by the composition of the tasks and dependencies between them. We generate different types of DAGs consisting of 5 to 10 tasks of three types of tasks (`fft`, `conv` and `decoder`). A DAG can have a criticality level of 1 or 2, and the fraction of `crit=2` DAGs in the trace reflects the congestion in the environment. Each DAG's deadline is set as the critical path time (CPT). We generate 1,000 DAG traces for the three congestion scenarios (urban, semi-urban and rural) with `crit=2` DAG fractions of 50%, 20%, and 10% for urban, semi-urban and rural, respectively. We then evaluate these traces at varying DAG arrival rates (AV speeds) to determine the best QoM metrics.

Table 5.4: Timing and power profile of evaluated kernels on each PE-type.

Suite	Task Type	Execution Time			Power (mW)		
		CPU	GPU	ASIC	CPU	GPU	ASIC
Mini-ERA	2D Convolution	583*	349*	180*	634	2225	445
	Viterbi Decoder	1021*	20*	-	864	1228	-
	2D FFT	3193*	97*	4*	1036	6364	4
ADSuite / MAVBench	Object Detection	3531†	156†	96†	3654	467	28
	Object Tracking	1825†	17†	2†	5600	12790	590
	Localization	165†	95†	10†	6133	4457	22
	Mission Planning	1†	-	-	3534	-	-
	Motion Planning	8†	-	-	4222	-	-
ADSuite	Fusion	0.1†	-	-	505	-	-
MAVBench	Occupancy Map Gen + Point Cloud Gen	976†	761†	-	2995	3533	-
	Shortest Path Planner	1005†	379†	-	3302	3533	-
	Collision Check	1†	-	-	500	-	-
	Path Tracking	1†	-	-	501		
	Frontier Exploration	397†	-	-	5980	-	-

*in micro-seconds †in milli-seconds

5.3.4.2 Real-World Application Traces

ADSuite and MAVBench have kernel components that make up the end-to-end applications' control flow graphs (CFGs). To generate DAGs, we took the CFGs of both ADSuite and MAVBench and studied scenarios that can lead to the execution of different sets of kernels. Such scenarios can arise from the vehicle changing route and leading to the execution of the mission planning kernels, and so on. Using the set of kernels executed in the CFG for a particular scenario, we generated DAGs with varying deadlines and criticalities. For ADSuite as described in [62], we set the deadline of each critical path task to be 100 ms. As no such information is provided for the MAVBench applications, each DAG's deadline is set as the CPT. We again generate 1,000 DAG traces and choose the same `crit=2` DAG fractions for urban, semi-urban, rural scenarios.

5.3.5 Simulation Platform

To explore multiple AV workloads and flexible SoC configurations that are not offered by a fixed real-world system, AVSched is implemented on the STOMP open-source scheduler evaluation platform [103]. STOMP is a queue-based discrete-event simulator used for early-stage evaluation of task scheduling mechanisms in heterogeneous platforms. To evaluate our DAG-based schedulers, we augmented STOMP to accept DAG-based inputs, while using the underlying queue-based simulator to schedule ready tasks. We also added real-time parameters, such as deadlines and safety criticalities. We realistically model the simulated SoC by providing STOMP with the power and timing profile of tasks obtained from the Jetson TX1 platform (Table 5.4). STOMP also provides the flexibility to add a deviation to the execution time to account for contention on shared resources like memory and buses, which we added as described in Section 5.3.2.4.

5.4 Evaluation

In this section, we evaluate AVSched for both synthetic and real-world application traces derived from the AV benchmark suites: Mini-ERA [79], ADSuite [62] and MAVBench [70]. We also incrementally evaluate the features of AVSched and compare against state-of-the-art schedulers, namely 2lvl-EDF [58], CPATH [54], RHEFT [57] and ADS [59] in terms of the QoM metrics (Section 5.1.4) and overall PE utilization.

5.4.1 Offline Profiling

The offline timing profiles generated for key kernels of both the synthetic and real-world applications for representative input data sizes are shown in Table 5.4.

5.4.2 Single-Level Task Scheduler Evaluation

Using the synthetic traces and offline timing profile, we first evaluate for varying environmental conditions our task scheduler, TS, against a commonly used greedy non-blocking task scheduler that assigns tasks to the next available PE with least execution time, `TS-greedy` [104]. The QoM results for these schedulers are shown in Figure 5.7(a). We see that for all congestion levels, TS achieves 7.8% better mission time over `TS-greedy`. Although `TS-greedy` has better utilization due to the greedy algorithm, TS's improved mission performance along with its ability to improve utilization when combined with the rank update optimization of AVSched. We thus consider TS as our `Task-Sched` policy for AVSched.

5.4.3 AVSched Optimizations

AVSched introduces two-level schedulers, MS_{static} and $MS_{dynamic}$ along with multiple optimizations, with the use of `Meta-Sched`. We evaluate MS_{static} and $MS_{dynamic}$ with these optimizations (hetero-rank ordering, hierarchical ranking and rank update) incrementally to analyze the benefits from each, under varying congestion levels. These features are evaluated with the DAG pruning optimization, *i.e.*, AVSched prunes non-critical DAGs that are unlikely to meet deadlines to reduce task traffic.

5.4.3.1 Two-level schedulers

Mission time improvement of the two types of two-level scheduling policies (MS_{static} and $MS_{dynamic}$ along with the pruning optimization) over TS for different environments is shown in Figure 5.7(b). MS_{static} and $MS_{dynamic}$ achieve speedups of up to 2.5× and 2.6× over TS, respectively.

5.4.3.2 Hetero- and Hierarchical Ranking

Employing hetero-ranking for MS_{static} and $MS_{dynamic}$ can help achieve a further reduction in mission time by up to 1.2× for both, as shown in Figure 5.7(c). Applying the hierarchical ranking feature over and above the hetero-ranking helps to improve mission performance by up to 1.4× and 2.1× for MS_{static} and $MS_{dynamic}$, respectively, as shown in Figure 5.7(d). These optimizations have negligible effect on utilization as they all use TS without load-balancing of non-critical tasks to slow PEs as the underlying task scheduler.

5.4.3.3 Rank Update and Load Balancing

AVSched enables a rank update optimization before a task is assigned to a PE to help improve the scheduling decision for tasks that have been waiting in the ready queue and whose deadlines are approaching. Figure 5.7(e) shows the benefits of additionally using the rank update feature over the two-level schedulers MS_{static} and $MS_{dynamic}$ with hierarchical ranking. Mission time is improved by 1.6× and 1.5×, respectively. Since both MS_{static} and $MS_{dynamic}$ are dependent on the sub-deadline of the task, updating the ranks help prioritize tasks that have been waiting in the ready queue while their deadlines approach. Moreover, TS's ability to balance slow PE utilization with non-critical tasks helps improve utilization by 2.3× and 2.1× for MS_{static} and $MS_{dynamic}$, respectively.

5.4.4 Scheduler Evaluation for Real-World Applications

For each of the real-world applications, we compare AVSched against prior baseline schedulers (CPATH, RHEFT, 2lvl-EDF and ADS) in terms of QoM metrics and PE utilization.

5.4.4.1 QoM Metrics Comparison

For ADSuite (Figure 5.8(a)), AVSched with MS_{static} achieves 1.8-10.1×, improvement in mission time for Sys_B, and 1.9-7.1× improvement in mission time for Sys_C, over the

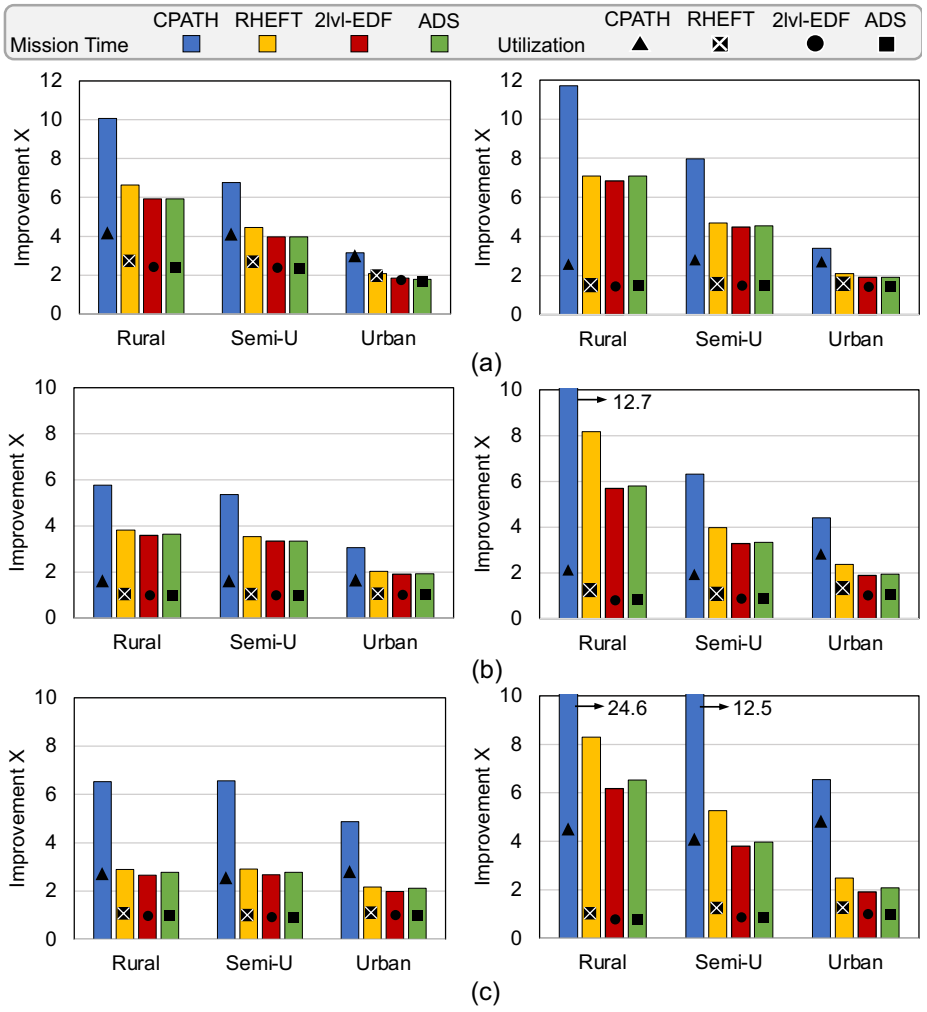


Figure 5.8: Comparison of mission performance and PE utilization of AVSched against prior-work schedulers for (a) ADSuite, (b) 3D Mapping, and (c) Package Delivery. *Left:* Sys_B. *Right:* Sys_C.

baseline schedulers. In terms of % mission completed, for Sys_B (not shown), the state-of-the-art schedulers complete just 38%, 5% and 7% of the mission at the maximum safe speed of AVSched for the rural, semi-urban and urban scenarios, respectively, before missing the deadline for a critical DAG. Furthermore, AVSched can achieve 1.7-4.2× and 1.4-2.8× better PE utilization over the baselines for Sys_B and Sys_C, respectively.

AVSched with $MS_{dynamic}$ achieves up to 1.9-5.8× improvement in mission time for Sys_B, and 2.0-12.7× improvement in mission time for Sys_C, over the state-of-the art schedulers, for 3D Mapping as shown in Figure 5.8(b). For the mission completed metric for

Sys_B, at the maximum safe speed achieved by AVSched, ADS and 2lvl-EDF are each able to complete a maximum of 9% of the mission. Moreover, in terms of utilization, AVSched achieves up to 1.7× and 2.8× better PE utilization for Sys_B and Sys_C, respectively. This improvement is lower than that for ADSuite, because 3D Mapping has a high CPU utilization and low accelerator and GPU utilization, since many tasks execute only on the CPU.

The mission time for Package Delivery is shown in Figure 5.8(c). AVSched with $MS_{dynamic}$ achieves 2.0-6.6× improvement in mission time for Sys_B, and up to 2.1-24.6 improvement in mission time for Sys_C, over the baseline schedulers. In terms of mission completion (not shown), ADS achieves a maximum of 38% mission at the maximum safe speed of AVSched for Sys_B. AVSched achieves up to 3.2× better average PE utilization for Sys_B.

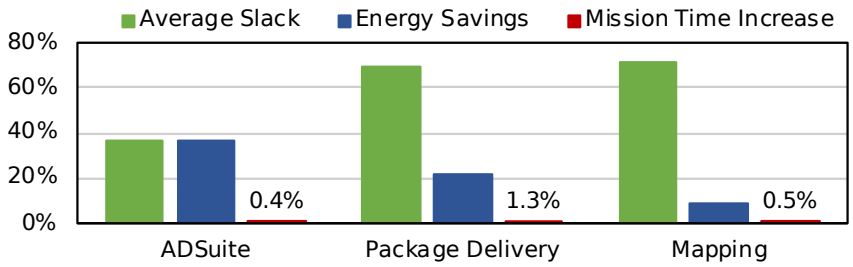
Note that many of the tasks executed in 3D Mapping and Package Delivery only have CPU implementations (Table 5.4). As developers produce heterogeneous versions of these tasks, we expect AVSched to exhibit higher improvements in terms of QoM metrics and PE utilization over the baseline schedulers. Moreover, AVSched with $MS_{dynamic}$ performs significantly better for ADSuite in comparison to the baseline schedulers, as the deadlines for this application are more stringent (∼400 ms) in comparison to MAVBench (∼2000 ms).

5.4.4.2 Scheduler Overhead

We also evaluated the overhead of AVSched, *i.e.*, the time spent for dependency tracking, meta information update, task prioritization and task assignment, running on the host Arm processor on the TX1. We observed this overhead to be no more than 19% and 6% of the total mission time for ADSuite and MAVBench, respectively.

5.4.4.3 Energy and Available Slack

As discussed in Section 5.3.3, we adopt a scheme that dynamically recuperates a fraction (f_{slack}) of the slack savings enabled by AVSched on a per-task basis (Section 5.3.3). Fig-



Slack + energy savings and mission time overhead with DVFS at highest AV speed

Application	Utilization %	CPU	CPU	CPU	CPU	CPU	CPU	CPU	CPU	GPU	GPU	Acc	Acc	Acc
ADSuite	No DVFS	93	92	89	85	87	82	11	1	91	84	47	99	12
	With DVFS	96	96	95	91	89	85	10	0	91	86	46	99	12
Package	No DVFS	99	95	97	97	96	94	91	88	98	95	24	0	0
	With DVFS	93	94	91	93	92	91	93	89	99	97	23	0	0
Mapping	No DVFS	98	98	98	98	97	95	96	95	94	94	29	0	0
	With DVFS	98	98	96	95	95	96	96	95	97	93	28	0	0

Per-PE utilization with and without DVFS at highest AV speed

Figure 5.9: *Top.* Slack, energy savings and mission overhead for AVSched with DVFS enabled across the three applications. *Bottom.* Per-PE utilization for each application. Results are shown for the rural congestion. The execution run corresponding to the f_{slack} value with best energy savings is reported here. Note that the utilization values for two accelerators (DET and TRA) are zero because the latter two applications do not have any tasks that use these PEs.

ure 5.9 (top) shows that our DVFS policy allows AVSched to achieve energy savings of 36%, 22% and 8% for ADSuite, 3D Mapping and Package Delivery, respectively, while increasing the mission time by just 0.4-1.3%. Note that this is at the maximum safe speed of the AV. At 85% of the AV's maximum safe speed, we observe energy savings of up to 46.6% (average PE utilization improvement of up to 18.9%) (not shown).

We also show the per-PE utilization with and without DVFS in Figure 5.9 (bottom), for each of the three applications. Much of the overall energy savings comes from DVFS on the CPU cores since the CPU is typically the slowest and consumes the most power (Table 5.4). DVFS improves the utilization of CPUs by 4% on average for ADSuite, yielding the highest energy savings among the applications. DVFS may also reduce the utilization of a subset of the slower cores whenever there are changes to the schedule such that tasks that previously were executed on these PEs now migrate to a faster PE. We observe this in the context of

Package Delivery and Mapping, where migration of tasks from CPU to GPU contributes to reduced utilization for the slower CPUs, and yet lowers the overall energy consumption.

5.4.4.4 SoC Design Optimization

As described in the Introduction of Chapter 5, having a scheduler-in-the-loop enables design space exploration to determine the best architectural configuration and level-of-heterogeneity for a given AV application. We used AVSched to determine the best SoC configuration that optimizes upon mission time, energy consumption and PE utilization when executing the three AV applications evaluated in this work. We explore a set of design points in Figure 5.10 and pick the best SoC configuration as the one with the minimum energy–mission time product and smallest number of PEs (best PE utilization). Across ADSuite, 3D Mapping and Package Delivery, the best SoC configurations are $(8, 1, 1, 2, 2)$, $(0, 0, 2, 4, 8)$ and $(0, 0, 2, 4, 8)$, respectively, where (A, B, C, D, E) denotes A detection accelerators, B tracking accelerators, C localization accelerators, D GPUs and E CPU cores.

5.5 Related Work

Autonomous vehicles pose the challenge to execute highly heterogeneous applications within stringent real-time and safety constraints. Prior work proposes the use of heterogeneous SoCs to help meet the performance constraints of individual tasks within the heterogeneous applications [52, 62]. However, during runtime, multiple critical applications and tasks are required to be executed simultaneously within their deadlines [105].

A plethora of work exists on scheduling algorithms for heterogeneous systems. Much of the prominent schedulers focus on optimizing the makespan, *i.e.*, execution time of a single DAG [53, 54, 106]. Tong *et al.* use Q-learning along with heterogeneous earliest finish time (HEFT) algorithm from [53] to reduce the makespan of a DAG [107]. Shetti *et al.* propose HEFT-NC [108] to optimize ranking and task selection of HEFT [53] by considering global

		ADSuite			3D Mapping			Package Delivery		
Acc	GPU	CPU Core								
		2	4	8	2	4	8	2	4	8
0	2	19.4	91.6	116	4.1	2.9	1.8	3.3	2.6	1.6
	4	7.8	29.3	54.8	2.5	2.4	1.5	2.1	3.3	1.3
	8	4.1	11.4	23.6	2.4	2.0	1.2	3.5	3.7	1.2
2	2	3.8	20.5	46.1	2.7	2.0	1.2	2.4	1.7	1.1
	4	3.2	12.1	27.5	1.8	1.4	1.0	1.5	1.2	1.0
	8	2.5	6.6	14.4	1.7	1.6	1.5	2.4	1.9	1.3
4	2	2.0	9.5	22.6	2.7	2.0	1.2	2.4	1.7	1.1
	4	1.9	7.0	16.7	1.8	1.4	1.0	1.5	1.3	1.0
	8	1.7	4.8	11.1	1.6	1.7	1.5	2.8	2.4	1.1
8	2	1.0	4.2	10.2	2.7	2.0	1.2	2.4	1.7	1.1
	4	1.0	3.2	7.7	1.8	1.4	1.0	1.5	1.2	1.0
	8	1.0	3.3	7.8	1.6	1.7	1.5	2.7	1.9	1.1

Figure 5.10: Normalized product of energy, mission time for varying SoC configurations for ADSuite, 3D Mapping and Package Delivery using AVSched in an urban scenario. We evaluate AVSched on SoC configurations that have varying CPU core count, GPU count, detection accel. count (for ADSuite – other PE types are at one count) and localization accel. count (for 3D Mapping and Package Delivery – other PE types are at zero count) to achieve the SoC configuration with the best energy and mission time (denoted in green).

and local processor information. However, as these schedulers are not optimized for the real-time requirements of the AVs and are not built for multiple DAG execution, they would need to be operated on AVs running at very low speeds to meet deadlines for all the critical DAGs of the mission.

To schedule for a multi-DAG scenario on heterogeneous systems, Xu *et al.* develop the reverse HEFT scheduling algorithm [57]. However, this algorithm is not feasible for dynamic systems as it requires *a priori* knowledge of arrival times of all DAGs. Real-time schedulers like earliest deadline first (EDF) and deadline monotonic (DM) cater to the needs of a real-time system where all tasks have a fixed priority, and the criticality of tasks is not considered [105]. However, AVs are categorized as cyber-physical systems and require schedulers that can schedule for mixed criticalities and multiple DAGs [109]. In this regard, Xie *et al.* [59] propose two dynamic schedulers; fairness-based dynamic

scheduler (FDS_MIMF) and an adaptive dynamic scheduler (ADS_MIMF), to optimize the makespan of the DAGs and to achieve low deadline miss ratio (DMR) by considering safety and criticality of the system values for high-criticality DAGs, respectively. Wu and Ryu [58], present the best speed fit EDF scheduler that prioritizes tasks according to the earliest deadline and assigns the task to the best possible PE while considering the execution profile of the task, similar to the 2lvl-EDF implementation in this work. However, none of these work consider that meeting real-time deadlines does not translate to safe completion of mission at the least mission time, *i.e.*, while operating the AV at the maximum safe speed. Moreover, use of HEFT-like algorithms for task scheduling on a highly heterogeneous SoC leads to low utilization in slow PEs. AVSched caters to both the requirements of an AV *i.e.*, to meet real-time deadlines for critical DAGs and reduce overall mission time.

5.6 Conclusion

Heterogeneous SoCs for autonomous vehicles (AVs), while necessary to meet stringent performance and safety constraints, pose challenges for traditional task scheduling approaches.

In this chapter, we presented AVSched, a multi-level scheduler that exploits the highly heterogeneous nature of the underlying SoC in conjunction with the characteristics of an AV application. AVSched's goal is to improve a global objective function, exemplified by a defined *Quality-of-Mission* (QoM) metric, providing a more *holistic* scheduling approach that looks into the full hardware-software AV stack to improve the overall mission's quality rather than focusing solely on the real-time requirements of individual kernels or applications. Our evaluation shows that AVSched improves overall mission performance by an average of **5.4**×, **3.2**×, **2.9**× and **2.9**× when compared to CPATH, RHEFT, 2lvl-EDF and ADS (current, state-of-the-art real-time heterogeneous schedulers). Furthermore, AVSched achieves an average of **52.4%** higher hardware utilization, while meeting **100%** of critical deadlines on real-world applications of autonomous driving and aerial vehicles. Collaterally, AVSched

can maximize the *slack* of individual kernels and applications, which can be exploited to improve the power-performance efficiency of the SoC through means such as dynamic voltage-frequency scaling (DVFS), without degrading the *Quality-of-Mission* metrics.

CHAPTER 6

Device-level Heterogeneous System

6.1 Introduction

Albeit manufacturing issues in emerging technologies exist, emerging devices can be utilized efficiently if designed to run suitable applications. Carbon nanotube transistors (CNFETs) and tunneling-based transistors (TFETs) have energy-delay and energy advantages in the near-threshold voltage region. 2D material transistors such as MoS2 FETs have lower leakage current than Si FETs. Emerging technologies such as Ge nanowire, carbon nanotube, and 2D materials have low process temperature, which enables monolithic 3D integration. This leads to the design of device-level heterogeneous systems with various emerging devices. To balance performance and energy of the system while taking advantage of the various emerging devices' properties, co-optimization in device level and architecture level is necessary.

Previous studies about configuring heterogeneous hardware with emerging devices showed the benefits of co-optimization at the architecture, transistor, and memory cell levels. CMOS-TFET hybrid multicore processor was investigated by adopting heterogeneous thread-to-core mapping, dynamic work partitioning, dynamic power partitioning, and application-to-core mapping [110, 111]. Inside a single CPU and GPU core, configuring some of the components with TFETs provided further EDP benefits [112]. Monolithic 3D integration of CNFETs, 2D materials, RRAM, and STT-MRAM achieved 1000× energy-delay product

improvement in abundant data computing [113].

We would like to propose a heterogeneous hardware system that can further utilize the benefits of co-optimization in various devices and heterogeneous architecture. To design an efficient heterogeneous hardware system, we propose the following features:

- Design of heterogeneous systems with multiple accelerators with various emerging devices. Depending on the applications, each processing element consists of different configurations of different emerging devices.
- Optimization for different workloads and multiple domains like communication, vision, autonomous vehicle application, mobile computing, and server application.

6.2 Background and Motivation

6.2.1 Task Requirements

Along with catering to the growing heterogeneity in the applications and kernels, modern systems must also cater to varying real-time and energy requirements. We consider the kernels to arrive with three types of task requirements;

`Type-D`: Tasks with a high-performance requirement

`Type-E`: Tasks with a low energy requirement

`Type-ED`: Tasks that need to operate at the optimal EDP point.

6.2.2 Emerging Compute Technologies

Dennard scaling and Moore's law have led the innovation of the semiconductor industry for decades. However, the Si CMOS technology is approaching its physical and structural limits. Beyond Si transistors such as carbon nanotube field-effect transistors (CNFETs) and tunneling field-effect transistors (TFETs) are candidates that can provide the breakthrough to the Si CMOS technology's limitations. The unique properties and advantages of these

emerging technologies will benefit the power and speed of the hardware system while allowing for scaling of transistor size, density, and geometry. Furthermore, the device-level heterogeneous system of the beyond Si transistors and Si CMOS can provide additional opportunity for metric optimization by exploiting each transistors' strengths.

6.2.2.1 Carbon Nanotube Field-Effect Transistor (CNFET)

CNFETs have a cylindrical rolled-up structure of a carbon hexagonal layer as a one-dimensional nanowire channel [6]. CNFETs' high mobility and gate-all-around structure lead CNFET processors to have better energy-delay product (EDP) than Si CMOS processors [6]. This makes CNFETs a strong candidate for low power, low supply voltage, near-threshold, and highly energy-efficient computing [11]. The low process temperature enables CNFETs to be adopted for monolithic 3D integration. However, the immaturity of the manufacturing process generates yield issues and density variations [6]. The high contact resistance of the CNFETs makes them have inferior performance than Si CMOS at high supply voltage [10].

6.2.2.2 Tunnel Field-Effect Transistor (TFET)

TFETs have a steep subthreshold slope because of their band-to-band tunneling current-conducting mechanism [6]. Along with Si, III-V materials such as GaSb and InAs are also used to manufacture TFETs. The low off-state leakage current, low threshold voltage, and low supply voltage make TFETs suitable for low power applications. TFETs can be integrated with the Si CMOS, which show the opportunity for fine-grained TFET-Si CMOS heterogeneous processors [112]. However, TFETs cannot replace Si CMOS in the high-performance application domain because of their low on-state current at high supply voltage [112].

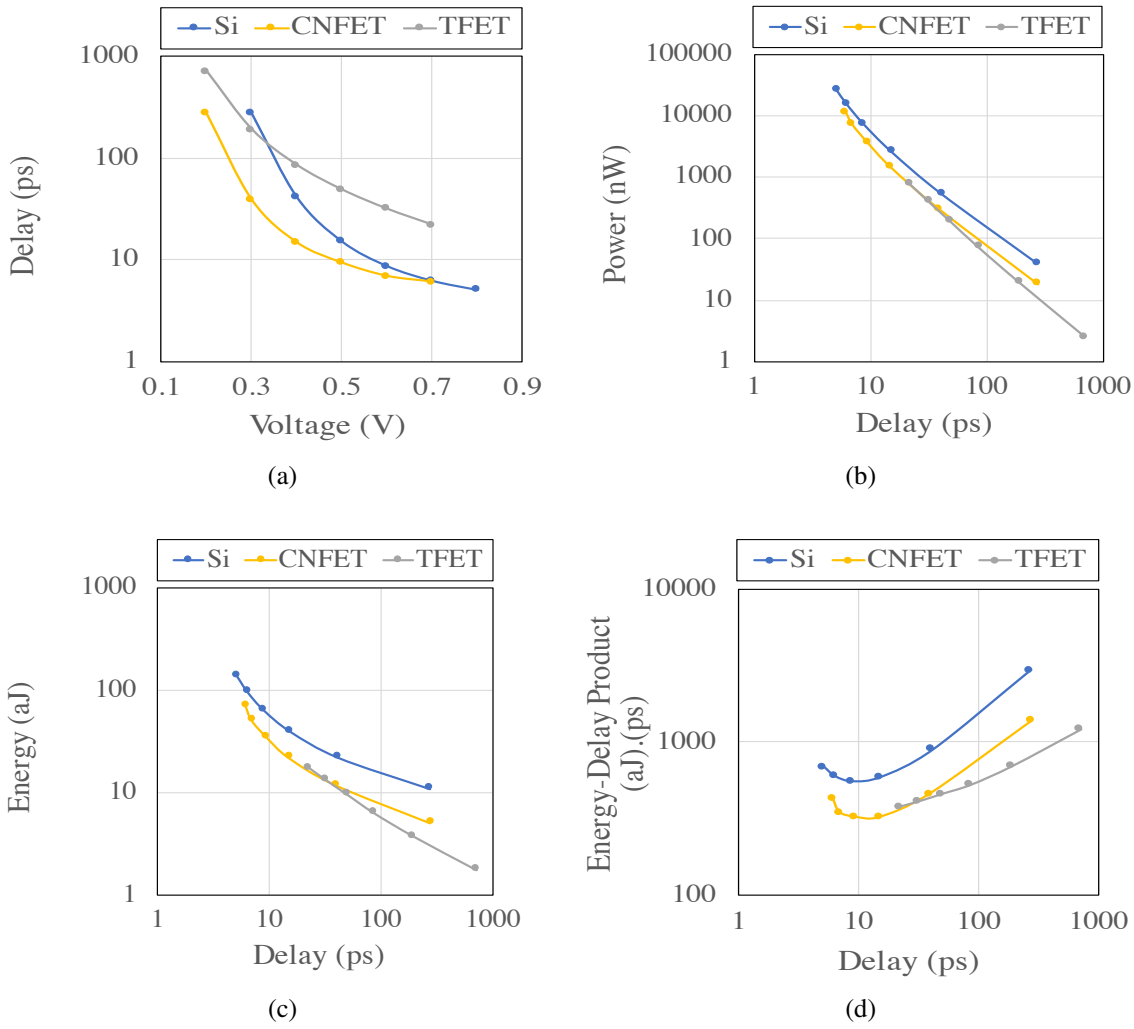


Figure 6.1: Comparison of FO4 inverter in Si, CNFET and TFET. We evaluate (a) delay across all operable voltages, (b) the trade-off between power and delay, (c) trade-off between energy and delay and (d) trade-off between energy-delay product and delay.

6.2.3 FO4 Inverter characterization

To understand the strengths of each technology, we profile FO4 inverters for Si, CNFET and TFET. Each FO4 inverter is built with a channel length of 14 nm and minimum width possible per technology.

As shown in Figure 6.1(a), Si-based FO4 inverter achieves the smallest delay at 0.8 V. However, as the operating voltage is reduced CNFET-based FO4 inverter has a lower delay than the Si-based FO4 inverter. Since, CNFET and TFET have lower threshold voltages,

they can operate at smaller voltages (0.2 V) than possible with Si. TFET starts to outperform Si in terms of delay at voltages below 0.3 V.

When we analyze the trade-off of delay with power and energy in Figure 6.1(b,c), although Si can achieve the highest performance, it also consumes higher power and energy. For a small trade-off in delay, CNFET can outperform Si in terms of power dissipated and energy consumed. However, TFETs can operate with very low power dissipation and energy consumption, emerging as low-power and low-energy devices. CNFETs on the other hand are optimized for both delay and energy and therefore are devices with best energy-delay product as shown in Figure 6.1(d).

Therefore, we can build a system using the three technologies to cater to tasks with requirements such as high performance, low power/energy, and least energy-delay product.

6.3 System Description

6.3.1 SoC Design

With the imminent "end" of Moore's Law, recent years have witnessed a surge of highly heterogeneous computing platforms composed of specialized accelerator units. This trend is also driven by the heterogeneity of the workloads that execute on those computing platforms, which come hand in hand with performance, efficiency and reliability constraints pertaining to specific application domains. While most heterogeneous systems have catered to the requirements of the application through architectural solutions, we can provide better scalability with the introduction of new technologies into the heterogeneous system. We call such systems as device-level, function-level heterogeneous systems. Moreover, due to the compatibility of the manufacturing process of CNFETs and TFETs with the Si-CMOS technology [110, 113], we can build efficient device-level heterogeneous SoCs that can take advantage of lower data movement costs through modern SoC interfaces.

For our work, we build a system with accelerators for three kernels from MachSuite [114],

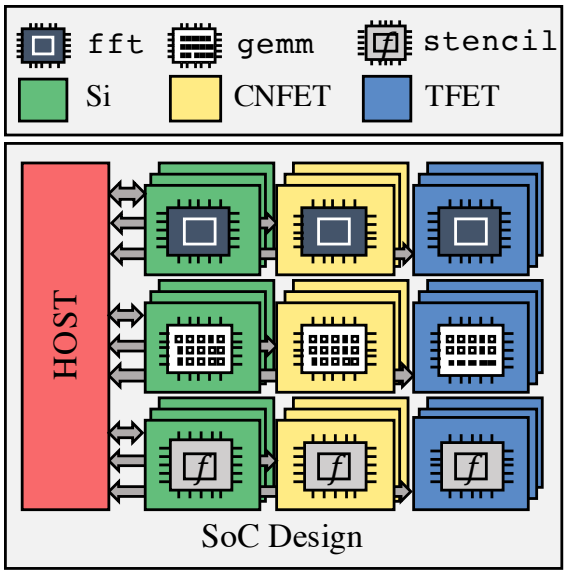


Figure 6.2: System Composition.

floating-point `fft`, integer `gemm` and integer `stencil`. Furthermore, as shown in Figure 6.2 each accelerator is also designed using the three technologies of Si, TFET and CNFET to cater to `type-D`, `type-E` and `type-ED` tasks. Using the methodology described in Chapter 5, we determine the number of accelerators of each kernel and device type, using the task characteristics, requirements and arrival rate. We also use AXI-based DMA interfaced accelerators as described in Shao *et al.* [115].

6.3.2 Scheduling Policy

We introduce a heterogeneous scheduling mechanism to address the added complexity introduced with each technology catering to a different task requirement.

First, the task scheduler reorder tasks in the ready queue according to the tasks' arrival time and the tasks' priority. We assign the highest priority to `type-D` tasks, followed by `type-ED` tasks and `type-E` tasks.

Each task at the head of the ready queue is then assigned to the PE with fastest completion time for `type-D` tasks. `Type-E` tasks are assigned to the PE with the smallest $Power \times Fastest\ Completion\ Time$. Similarly, `type-ED` tasks are assigned to the PE with the

smallest $Power \times Fastest\ Completion\ Time^2$.

Moreover, to save on leakage energy caused due to the steep slope devices, we introduce a scheduler driven, sleep-based power optimization. When no task is scheduled on the CNFET and TFET-based accelerators, the scheduler provides hints to the power management system to power gate the accelerator when it is idle for time $T_{threshold}$, and power it back on after time T_{sleep}.

6.4 Methodology

6.4.1 Spice Simulation

We first build standard cell libraries of CNFET and TFET by referencing the GF 12nm standard cell library using the methodology from Amarnath *et al.* [10]. Following this, we perform HSpice simulation to characterize the power and delay of each technologies' digital circuit components. The list of the components is 2-bit logical xor gate, and gate, or gate, 32-bit integer adder, 32-bit integer adder, 32-bit left shifter, 32-bit right shifter, register, single-precision 3-stage pipelined floating-point adder and multiplier. We feed the characterized power and delay numbers into the gem5-Aladdin framework [115] to obtain accelerator characteristics through a design space exploration done for each accelerator.

6.4.2 Accelerator Characterization

Gem5-Aladdin [115] is a pre-RTL performance and power modeling tool that enables rapid design-space exploration of accelerator designs. Among the parameters varied are the degree of loop unrolling, number of ports connecting to local memory (*i.e.*, scratchpad) and the partitioning scheme for individual data structures. We augmented gem5-Aladdin to explore additional design points by selecting different circuit-level designs based on the device type, voltage, and clock period. The latencies for clocked modules are considered as the

input clock period times the number of pipeline stages (fixed to three, in our work). The combinational modules are appropriately pipelined as needed, since these can have critical path delays greater than the specified clock period.

We use gem5-Aladdin to generate accelerator designs for three kernels from MachSuite [114], a collection of kernels that are popular candidates for hardware accelerator. These are `fft/strided` (floating-point), `stencil/stencil2d` (integer) and `gemm/blocked` (integer). In order to curb the simulation time, we only perform gem5-Aladdin simulation for the (V_{DD}, f) that are Pareto-optimal on the voltage-frequency characterization curve. We record the power, energy and energy-delay product (EDP), in addition to the accelerator latency, for each of the kernels and device types.

6.4.3 Heterogeneous System Setup

Using the accelerator latency, power, energy and energy-delay product (EDP), we build a hetero-system with three accelerators for each type of kernel: `fft`, `gemm` and `stencil`. Additionally, using [116,117], we also build each accelerator using the different technologies of Si, TFET and CNFET to meet task requirements of high performance, low energy and optimal EDP, respectively. We compare this system against a baseline Si-system that uses DVFS for each accelerator to meet the high performance, low energy and optimal EDP requirements.

6.4.4 Task Traces

We build four different 1000-task traces based on the amount of `Type-D` and `Type-E` tasks in the trace along with 500 `Type-ED` tasks. Based on the number of `Type-D` tasks in the remaining 500 tasks, we consider four traces TR1 (20%), TR2 (40%), TR3 (60%) and TR4 (80%).

6.4.5 Metrics of Evaluation

To evaluate our systems, we consider multiple metrics of evaluations. For the gates, modules and standalone accelerators, we consider delay/latency, power, energy and energy-delay product (EDP). For the system, level evaluation, we additionally also compare the tasks' response time and overall execution time of the task traces.

6.5 Evaluation

6.5.1 Module Characterization

We present the characterization of 32-bit integer adder and multiplier and single precision floating-point 3-stage adder and multiplier for each of the three technologies of Si, CNFFT and TFET.

6.5.1.1 32-bit Integer Adder

The characterization of a 32-bit integer adder designed as a sparse tree adder for each of the three technologies is shown in Figure 6.3. Si is able to achieve the fastest module with a delay of 0.3 ns at 0.8 V. This is 27% and 4.9× faster than the CNFET and TFET modules operating at 0.7 V, respectively.

In terms of power and energy, TFET-based adder is the best with a power of 40 nW and energy consumption of 1.9 fJ. In comparison to CNFET and Si, TFET consumes 7× and 11.5× lower power and 1.6× and 2.5× lower energy, respectively.

Due to its moderate energy consumption and performance, CNFET achieves the best EDP which is 43% and 20% better than Si and TFET, respectively.

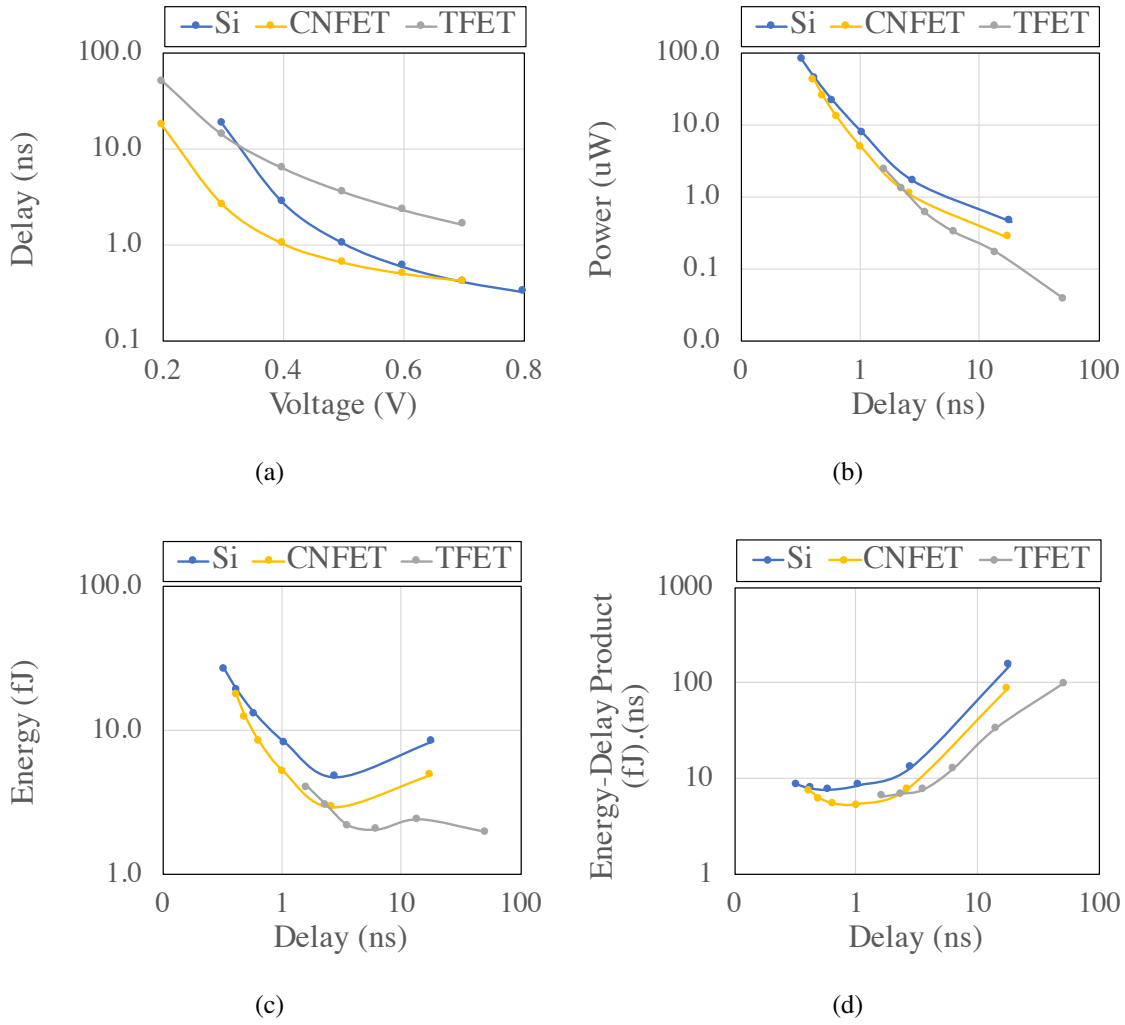


Figure 6.3: Comparison of 32-bit integer adder in Si, CNFET and TFET. We evaluate (a) delay across all operable voltages, (b) the trade-off between power and delay, (c) trade-off between energy and delay and (d) trade-off between energy-delay product and delay.

6.5.1.2 32-bit Integer Multiplier

The characterization of a 32-bit integer multiplier for each of the three technologies is shown in Figure 6.4. Similar to the 32-bit integer adder, Si is able to achieve the fastest module with a delay of 0.5 ns at 0.8 V. This is 55% and 5.4× faster than the CNFET and TFET modules operating at 0.7 V, respectively.

In terms of power and energy, TFET-based adder is the best with a power of 40 nW and energy consumption of 2.9 fJ. In comparison to CNFET and Si, TFET consumes 25×

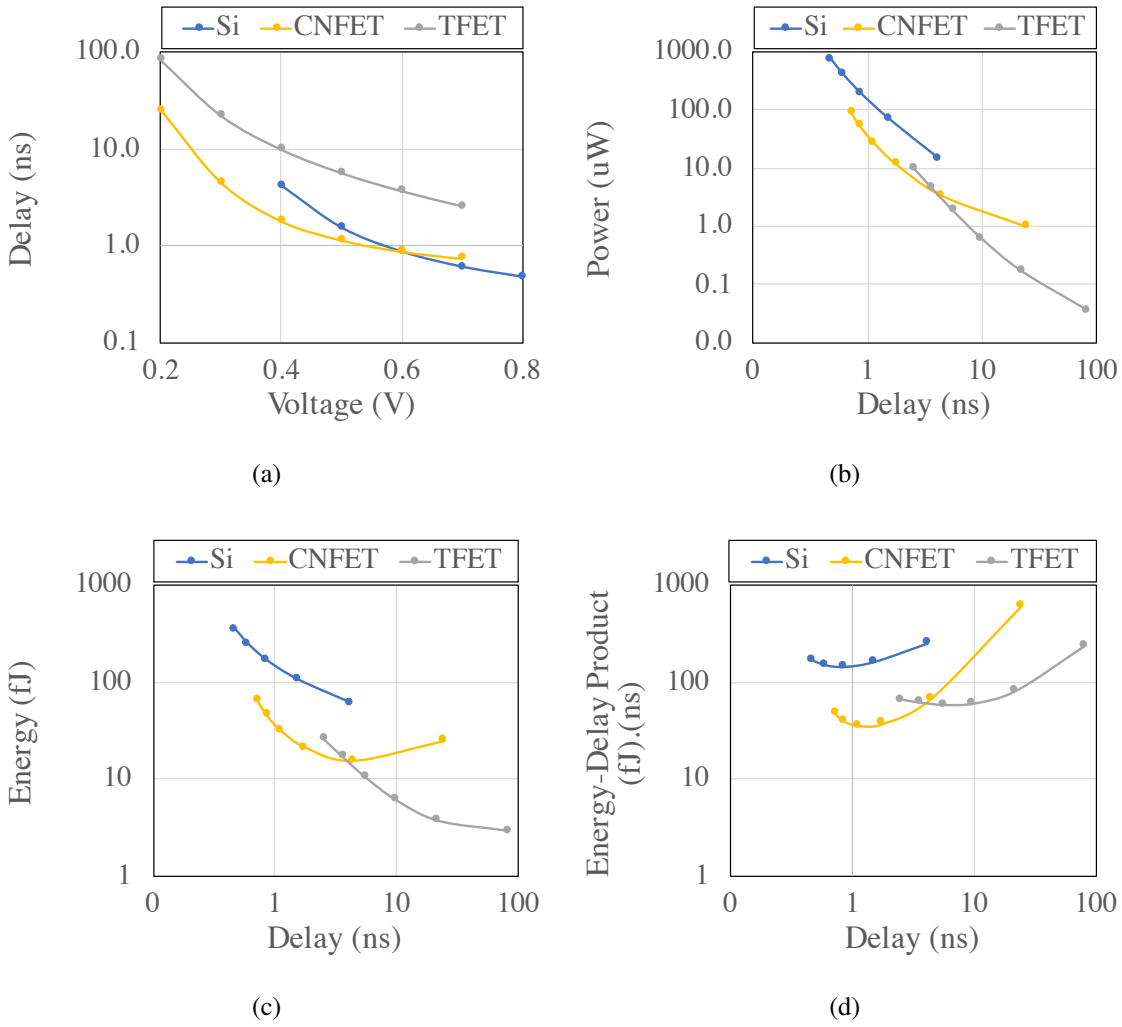


Figure 6.4: Comparison of 32-bit integer multiplier in Si, CNFET and TFET. We evaluate (a) delay across all operable voltages, (b) the trade-off between power and delay, (c) trade-off between energy and delay and (d) trade-off between EDP and delay.

and 360× lower power and 5.2× and 20.8× lower energy, respectively. CNFET achieves the best EDP which is 4× and 1.6× better than Si and TFET, respectively. Note that each technologies best point achieves better improvement in the case of the multiplier module over the adder module due to increased design and time complexity of the multiplier module.

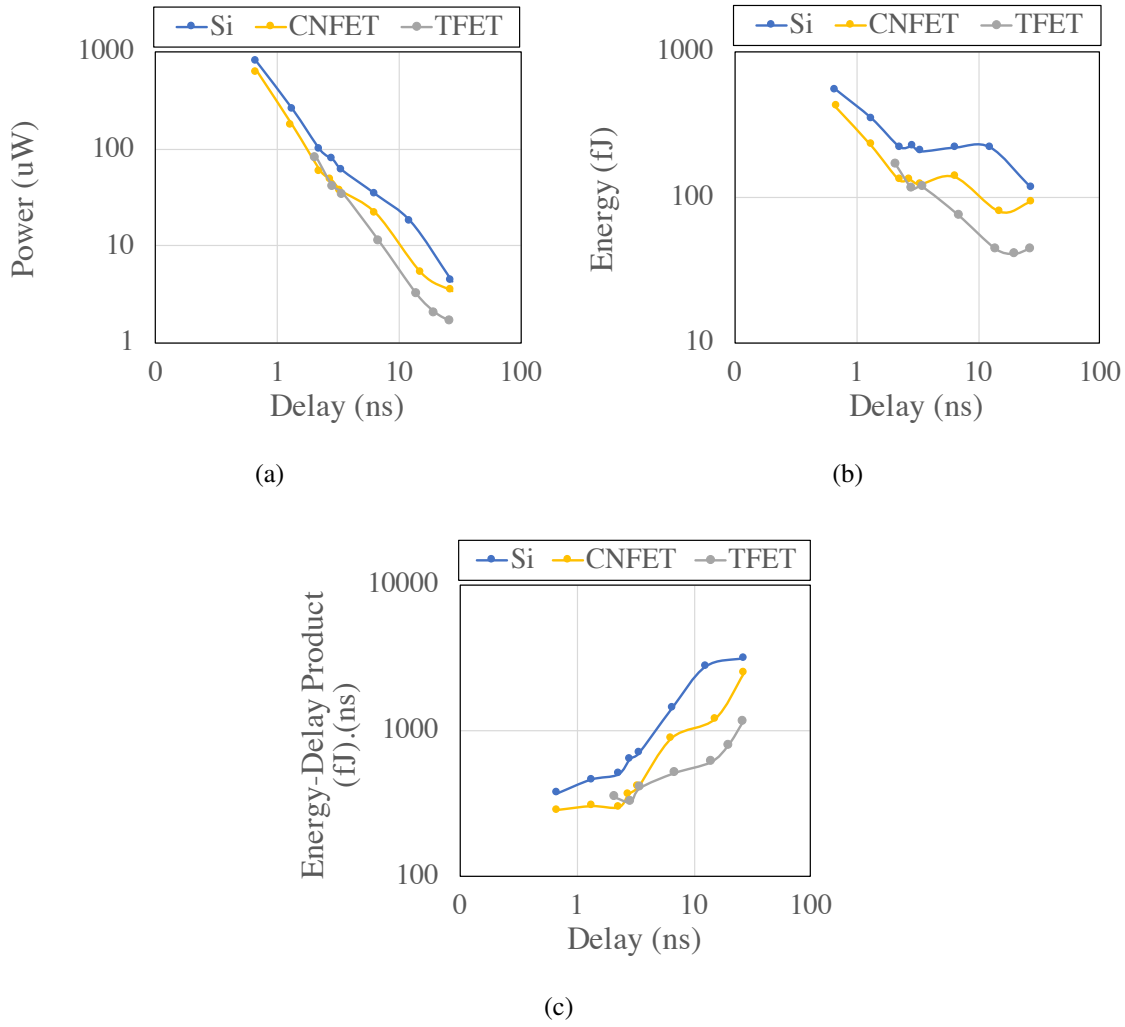


Figure 6.5: Comparison of single precision floating point adder in Si, CNFET and TFET. We evaluate (a) the trade-off between power and delay, (b) trade-off between energy and delay and (c) trade-off between EDP and delay.

6.5.1.3 Single Precision Floating-Point Adder

For the evaluation of the single precision floating-point three-stage adder, we study the trade-off of power, energy and energy-delay product against the adder latency. The results are shown in Figure 6.5. TFET achieves 2.6×, 2.1× lower power and 24×, 16.4× lower energy than Si and CNFET, respectively. CNFET achieves 11% and 15%lower EDP than Si and TFET-based adders.

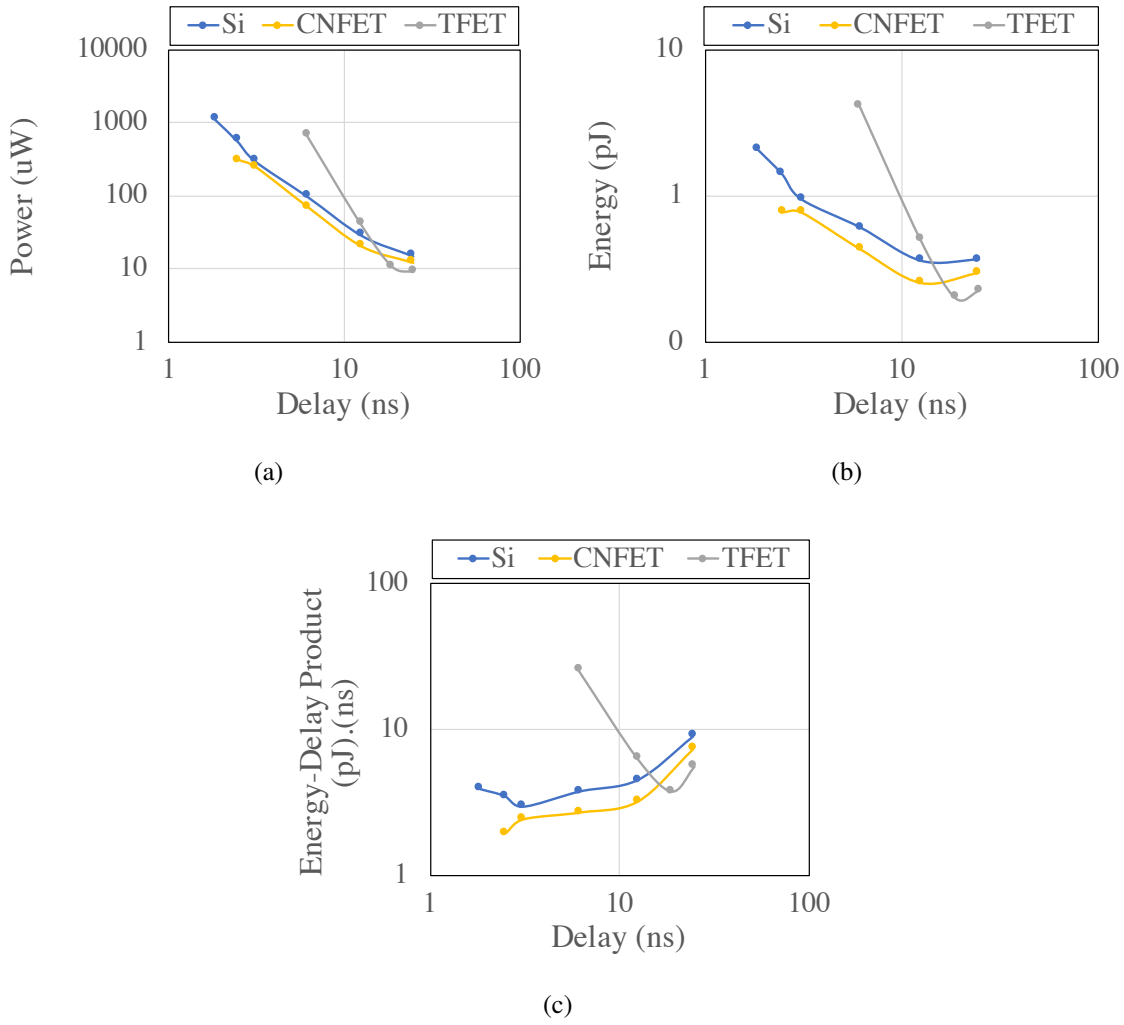


Figure 6.6: Comparison of single precision floating point multiplier in Si, CNFET and TFET. We evaluate (a) delay across all operable voltages, (b) the trade-off between power and delay, (c) trade-off between energy and delay and (d) trade-off between EDP and delay.

6.5.1.4 Single Precision Floating-Point Multiplier

The evaluation of the single precision floating-point three-stage multiplier is shown in Figure 6.6. TFET achieves 64% and 36% lower energy than Si and CNFET, respectively. CNFET achieves 39% and 96%lower EDP than Si and TFET-based multipliers.

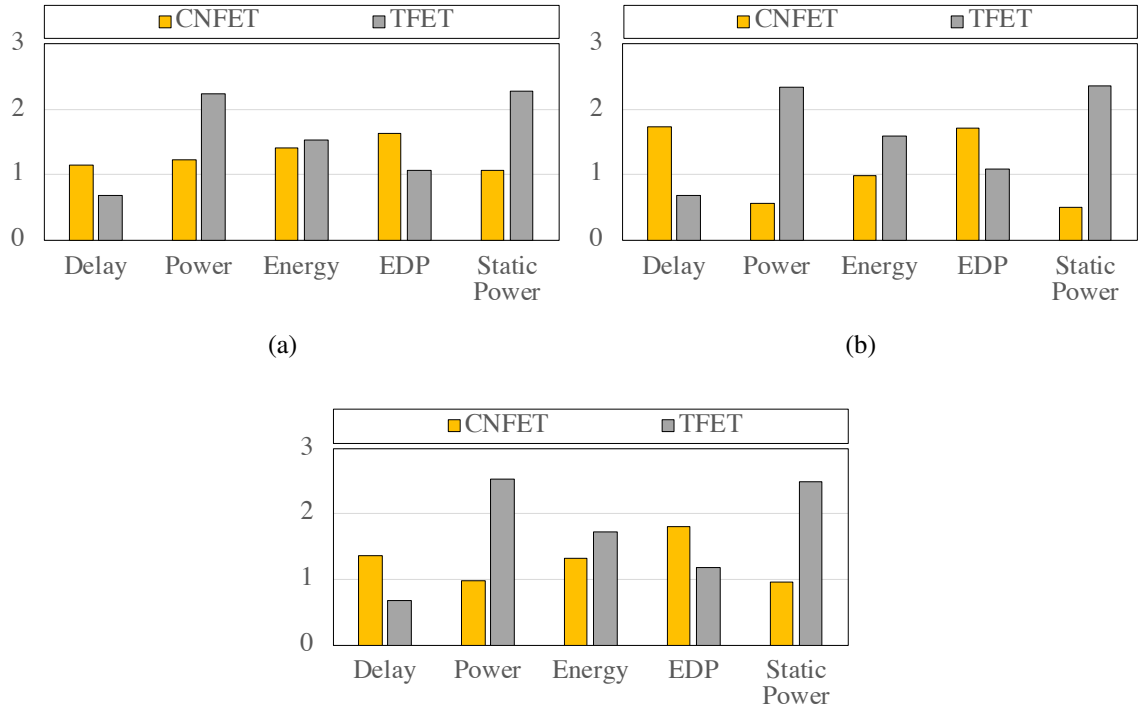


Figure 6.7: Comparison of metrics for (a) single precision `fft` accelerator, (b) 32-bit integer `gemm` and (c) 32-bit integer `stencil` accelerator. CNFET-based designs are compared against Si-based design operating at the best EDP. TFET-based designs are compared against Si-based design operating at the best energy.

6.5.2 Accelerator Characterization

We compare CNFET-based and TFET-based accelerators against Si-based design operating at the best EDP and the best energy, respectively. The results for delay, power, energy and EDP of each accelerator are shown in Figure 6.7. TFET achieves an energy improvement of up to 1.7×. CNFET achieves an EDP improvement of up to 1.8×.

6.5.3 System Evaluation

We compare the device-level, function-level heterogeneous system built with accelerators for `fft, gemm` and `stencil` using Si, CNFET and TFET technologies (S_{hetero}) against the baseline function-level heterogeneous system in Si with DVFS enabled to operate at

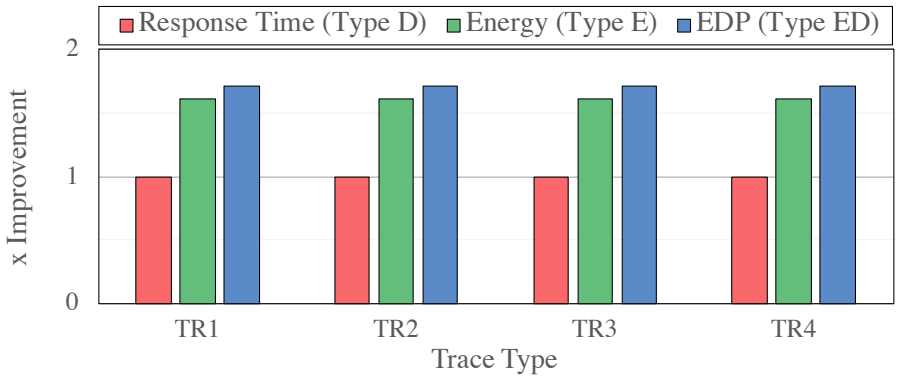


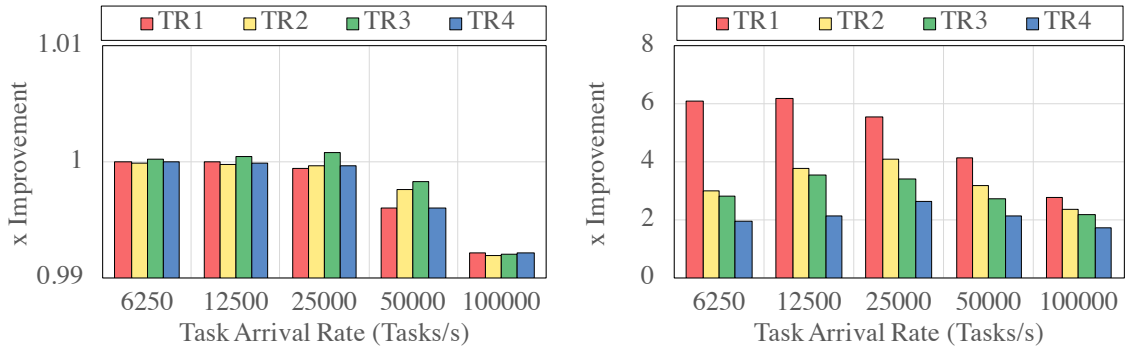
Figure 6.8: Comparison of response time for `Type-D` tasks, energy for `Type-E` tasks and EDP for `Type-ED` tasks of hetero-system over baseline Si-system.

the best delay, energy and EDP operational points (S_{homo}). The results for improvement in response time for `Type-D` tasks, energy for `Type-E` tasks and EDP for `Type-ED` tasks are shown in Figure 6.8. We achieve the same performance as the S_{homo} system for the `task-D` tasks, while achieving 1.6× energy improvement and 1.7× EDP improvement for the `task-E` and `task-ED` tasks, respectively. Also note, that the benefits across the different types of tasks do not vary due to the fact that all the benefits observed are from the use of different technologies and the scheduling overheads are not included.

We also evaluate the system S_{hetero} with the detailed sleep optimization in comparison to the S_{homo} system without the sleep optimization as shown Figure 6.9. We observe that as the task arrival rate increases, there is a minimal degradation in execution time (<0.01%). However, S_{hetero} is able to achieve up to 6.2× for TR1 trace (maximum `Type-E tasks`) and up to 2.6× for TR4 trace (maximum `Type-D tasks`).

6.6 Conclusion

Heterogeneous SoCs are developed to cater to growing requirements of highly heterogeneous applications. Prior art has explored heterogeneity either at the function-level or the device-level. This work explores combining the two to cater to performance and energy



(a) (b)

Figure 6.9: Comparison of (a) total execution time and (b) energy consumption of the hetero-system over Si-system for varying task arrival rates. Note that the hetero-system employs fine grained power management to reduce idle leakage energy.

requirements of common kernels in both server and embedded system applications.

In this work, we presented a function-level and device-level heterogeneous SoC, S_{hetero}, built to accelerate kernels, like fast Fourier transform, general matrix multiplication and convolution-based stencil kernels, using three different device technologies of silicon FETs, carbon nanotube FETs and tunnel FETs. The goal of the work is to cater to performance, energy and energy-delay requirements of tasks using the different accelerators built using each device technology based on their operational strengths.

We show that S_{hetero} achieves 1.6× energy improvement and 1.7× EDP improvement for the `task-E` and `task-ED` tasks, respectively, while achieving the same performance for `task-D` tasks, over a homogeneous Si-based system with DVFS enabled. Furthermore, combining the system with a scheduler-driven sleep-based power optimization allows for a 1.7-6.2× improvement in system energy for varying task traces and arrival rates.

CHAPTER 7

Conclusion

This thesis develops solutions at multiple levels of the compute stack to help enable emerging technologies to be beneficially used in semiconductor products while mitigating variation and yield issues to continue performance scaling at reduced power consumption. At the device-level, we modeled variation observed in carbon-nanotube transistors (CNFETs) and studied its effect on performance and energy consumption. At the circuit-level, we proposed utilization of pass transistor logic for CNFET as an alternate to CMOS logic family to reap benefits of CNFETs shown through theoretical models. At the architecture-level, we proposed 3DTUBE, a yield enhancing, multi-granular reconfigurable 3D framework to improve performance in the presence of failures. At the system-level, we build a multi-accelerator and multi-technology heterogeneous systems. Lastly, at the operating system-level, we use efficient workload scheduling that optimize for varying performance and energy constraints of the heterogeneous systems.

BIBLIOGRAPHY

[1] Christensen, C., *The innovator's dilemma: when new technologies cause great firms to fail*, Harvard Business Review Press, 2013.

[2] Ajayi, T., Al-Hawaj, K., Amarnath, A., Dai, S., Davidson, S., Gao, P., Liu, G., Lotfi, A., Puscar, J., Rao, A., et al., "Celerity: An open source RISC-V tiered accelerator fabric," *Symp. on High Performance Chips (Hot Chips)*, 2017.

[3] Pal, S., Beaumont, J., Park, D.-H., Amarnath, A., Feng, S., Chakrabarti, C., Kim, H.-S., Blaauw, D., Mudge, T., and Dreslinski, R., "Outerspace: An outer product based sparse matrix multiplication accelerator," *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, 2018, pp. 724–736.

[4] Bagherzadeh, J., Amarnath, A., Tan, J., Pal, S., and Dreslinski, R. G., "R2D3: a reliability engine for 3D parallel systems," *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6.

[5] Pal, S., Feng, S., Park, D.-h., Kim, S., Amarnath, A., Yang, C.-S., He, X., Beaumont, J., May, K., Xiong, Y., Kaszyk, K., Morton, J. M., Sun, J., O'Boyle, M., Cole, M., Chakrabarti, C., Blaauw, D., Kim, H.-S., Mudge, T., and Dreslinski, R., "Transmuter: Bridging the Efficiency Gap Using Memory and Dataflow Reconfiguration," *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, PACT '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 175–190.

[6] Kim, H., Amarnath, A., Bagherzadeh, J., Talati, N., and Dreslinski, R. G., "A Survey Describing Beyond Si Transistors and Exploring Their Implications for Future Processors," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, Vol. 17, No. 3, 2021, pp. 1–44.

[7] ITRS, "International Technology Roadmap for Semiconductors," http://www.itrs.net/models.html, 2013.

[8] Franklin, A. D., Tulevski, G. S., Han, S.-J., Shahrjerdi, D., Cao, Q., Chen, H.-Y., Wong, H.-S. P., and Haensch, W., "Variability in carbon nanotube transistors: Improving device-to-device consistency," *ACS nano*, Vol. 6, No. 2, 2012, pp. 1109–1115.

[9] Zhang, J., Patil, N. P., Hazeghi, A., Wong, H.-S. P., and Mitra, S., "Characterization and design of logic circuits in the presence of carbon nanotube density variations," *TCAD*, Vol. 30, No. 8, 2011, pp. 1103–1113.

[10] Amarnath, A., Feng, S., Pal, S., Ajayi, T., Rovinski, A., and Dreslinski, R. G., "A carbon nanotube transistor based RISC-V processor using pass transistor logic," *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, IEEE, 2017, pp. 1–6.

[11] Amarnath, A., Bagherzadeh, J., Tan, J., and Dreslinski, R. G., "3DTUBE: A Design Framework for High-Variation Carbon Nanotube-based Transistor Technology," *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, July 2019, pp. 1–6.

[12] Amarnath, A., Pal, S., Kassa, H. T., Vega, A., Buyuktosunoglu, A., Franke, H., Wellman, J.-D., Dreslinski, R., and Bose, P., "Heterogeneity-Aware Scheduling on SoCs for Autonomous Vehicles," *IEEE Computer Architecture Letters*, 2021.

[13] Avouris, P., "Molecular Electronics with Carbon Nanotubes," *Acc. Chem. Res.*, 2002.

[14] Franklin, A. D., Luisier, M., Han, S. J., Tulevski, G., Breslin, C. M., Gignac, L., Lundstrom, M. S., and Haensch, W., "Sub-10 nm carbon nanotube transistor," *Nano Lett.*, 2012.

[15] Lee, C. S., Pop, E., Franklin, A. D., Haensch, W., and Wong, H. S. P., "A Compact Virtual-Source Model for Carbon Nanotube FETs in the Sub-10-nm Regime-Part I: Intrinsic Elements," *IEEE Trans. Electron Devices*, 2015.

[16] Avouris, P., Chen, Z., and Perebeinos, V., "Carbon-based electronics," *Nat. Nanotechnol.*, 2007.

[17] Wei, H., Shulaker, M., Wong, H. S. P., and Mitra, S., "Monolithic three-dimensional integration of carbon nanotube FET complementary logic circuits," *IEDM*, Dec 2013, pp. 19.7.1–19.7.4.

[18] Shulaker, M. M., Pitner, G., Hills, G., Giachino, M., Wong, H.-S. P., and Mitra, S., "High-performance carbon nanotube field-effect transistors," *2014 IEEE Int. Electron Devices Meet.*, 2014.

[19] Park, H., Afzali, A., Han, S.-J., Tulevski, G. S., Franklin, A. D., Tersoff, J., Hannon, J. B., and Haensch, W., "High-density integration of carbon nanotubes via chemical self-assembly," *Nat. Nanotechnol.*, 2012.

[20] Brady, G. J., Way, A. J., Safron, N. S., Evensen, H. T., Gopalan, P., and Arnold, M. S., "Quasi-ballistic carbon nanotube array transistors with current density exceeding Si and GaAs," *Science Advances*, Vol. 2, No. 9, 2016.

[21] Franklin, A. D. and Chen, Z., "Length scaling of carbon nanotube transistors." *Nat. Nanotechnol.*, 2010.

[22] "Online STANFORD Virtual Source - CNFET model," https://nano.stanford.edu/stanford-cnfet2-model.

[23] Bobba, S., Zhang, J., Gaillardon, P.-E., Wong, H.-S. P., Mitra, S., and Micheli, G. D., "System Level Benchmarking with Yield-Enhanced Standard Cell Library for Carbon Nanotube VLSI Circuits," *ACM JETC.*, 2014.

[24] Ding, L., Zhang, Z., Pei, T., Liang, S., Wang, S., Zhou, W., Liu, J., and Peng, L. M., "Carbon nanotube field-effect transistors for use as pass transistors in integrated logic gates and full subtractor circuits," *ACS Nano*, 2012.

[25] Patil, N., Deng, J., Mitra, S., and Wong, H. S. P., "Circuit-Level Performance Benchmarking and Scalability Analysis of Carbon Nanotube Transistor Circuits," *IEEE Transactions on Nanotechnology*, Vol. 8, No. 1, Jan 2009, pp. 37–45.

[26] Ghavami, B., Raji, M., Pedram, H., and Arjmand, O. N., "CNT-count Failure Characteristics of Carbon Nanotube FETs under Process Variations," *Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, Oct 2011, pp. 86–92.

[27] Wang, C., Sun, Y., Hu, S., Jiang, L., and Qian, W., "Variation-Aware Global Placement for Improving Timing-Yield of Carbon-Nanotube Field Effect Transistor Circuit," *ACM Trans. Des. Autom. Electron. Syst.*, Vol. 23, No. 4, June 2018, pp. 44:1–44:27.

[28] Ahmed, Z., Zhang, L., Sarfraz, K., and Chan, M., "Modeling CNTFET Performance Variation Due to Spatial Distribution of CNTs," *IEEE Trans. on Electron Devices*, Sept 2016, pp. 3776–3781.

[29] Zhang, J., Patil, N., Wong, H. S. P., and Mitra, S., "Overcoming CNT variations through co-optimized technology and circuit design," *IEDM*, Dec 2011, pp. 4.6.1–4.6.4.

[30] Hills, G., "Variation-Aware NDK," https://nanohub.org/resources/22582, Jul 2015.

[31] Schuster, S. E., "Multiple word/bit line redundancy for semiconductor memories," *JSSC*, Vol. 13, No. 5, 1978, pp. 698–703.

[32] Esmaeilzadeh, H., Blem, E., St Amant, R., Sankaralingam, K., and Burger, D., "Dark Silicon and the End of Multicore Scaling," *ISCA*, 2011, pp. 365–376.

[33] Cho, G., Kim, Y. B., Lombardi, F., and Choi, M., "Performance evaluation of CNFET-based logic gates," *I2MTC*, 2009.

[34] Kumar, K., Sahithi, C., Sahoo, R., and Sahoo, S. K., "Ultra Low Power Full Adder Circuit using Carbon Nanotube Field Effect Transistor," *ICPCES*, 2014.

[35] Shulaker, M. M., Hills, G., Patil, N., Wei, H., Chen, H.-Y., Wong, H.-S. P., and Mitra, S., "Carbon nanotube computer," *Nature*, 2013.

[36] Lee, Y., Ou, A., and Magyar, A., "Z-scale: Tiny 32-bit RISC-V Systems," *OpenRISC Conf.*, 2015.

[37] Waterman, A. S., *Design of the RISC-V Instruction Set Architecture*, Ph.D. thesis, EECS Department, University of California, Berkeley, 2016.

[38] Rabaey, J. M., Chandrakasan, A. P., and Nikolic, B., *Digital integrated circuits*, Prentice hall Englewood Cliffs, 1996.

[39] Lee, C. S., Pop, E., Franklin, A. D., Haensch, W., and Wong, H. S. P., "A Compact Virtual-Source Model for Carbon Nanotube FETs in the Sub-10-nm Regime-Part II: Extrinsic Elements, Performance Assessment, and Design Optimization," *IEEE Trans. Electron Devices*, 2015.

[40] Shulaker, M. M., Saraswat, K., Wong, H. S. P., and Mitra, S., "Monolithic three-dimensional integration of carbon nanotube FETs with silicon CMOS," *VLSI-T*, June 2014, pp. 1–2.

[41] Amarnath, A., Feng, S., Pal, S., Ajayi, T., Rovinski, A., and Dreslinski, R. G., "A carbon nanotube transistor based RISC-V processor using pass transistor logic," *ISLPED*, July 2017.

[42] "SkyWater Begins Work With MIT on Next-Generation Technology Development for DARPA Electronics Resurgence Initiative," 2018.

[43] Almudever, C. and Rubio, A., "Variability & reliability analysis of CNFET technology: Impact of manufacturing imperfections," *Microelectronics Reliability*, 2015, pp. 358–366.

[44] Zhang, J., Patil, N., Hazeghi, A., and Mitra, S., "Carbon Nanotube circuits in the presence of carbon nanotube density variations," *DAC*, July 2009, pp. 71–76.

[45] Blaauw, D., Chopra, K., Srivastava, A., and Scheffer, L., "Statistical Timing Analysis: From Basic Principles to State of the Art," *TCAD*, Vol. 27, No. 4, April 2008, pp. 589–607.

[46] Agarwal, A., Blaauw, D., Zolotov, V., Sundareswaran, S., Zhao, M., Gala, K., and Panda, R., "Path-based statistical timing analysis considering inter-and intra-die correlations," *TAU*, 2002, pp. 16–21.

[47] Gupta, S., Feng, S., Ansari, A., and Mahlke, S., "StageNet: A Reconfigurable Fabric for Constructing Dependable CMPs," *IEEE Trans. on Computers*, Vol. 60, No. 1, 2011.

[48] Bagherzadeh, J. and Bertacco, V., "3DFAR: A three-dimensional fabric for reliable multi-core processors," *DATE*, 2017.

[49] Wang, S., Tahoori, M. B., and Chakrabarty, K., "Defect clustering-aware spare-TSV allocation for 3D ICs," *ICCAD*, Nov 2015, pp. 307–314.

[50] Leon, A. S., Tam, K. W., Shin, J. L., Weisner, D., and Schumacher, F., "A Power-Efficient High-Throughput 32-Thread SPARC Processor," *JSSC*, Vol. 42, No. 1, Jan 2007, pp. 7–16.

[51] "NVIDIA Drive AGX Platform," https://developer.nvidia.com/drive.

[52] Talpes, E., Das Sarma, D., Venkataramanan, G., Bannon, P., McGee, B., Floering, B., Jalote, A., Hsiong, C., Arora, S., Gorti, A., and Sachdev, G. S., "Compute Solution for Tesla's Full Self Driving Computer," *IEEE Micro*, 2020, pp. 1–1.

[53] Topcuoglu, H., Hariri, S., and Wu, M.-y., "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, Vol. 13, No. 3, 2002, pp. 260–274.

[54] Chronaki, K., Rico, A., Casas, M., Moretó, M., Badia, R. M., Ayguadé, E., Labarta, J., and Valero, M., "Task scheduling techniques for asymmetric multi-core systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 7, 2016, pp. 2074–2087.

[55] "Earliest Deadline First Scheduling Algorithm." https://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling.

[56] "Fixed-priority Pre-emptive Scheduling." https://en.wikipedia.org/wiki/Fixed-priority_pre-emptive_scheduling.

[57] Xu, X.-J., Xiao, C.-B., Tian, G.-Z., and Sun, T., "Hybrid scheduling deadline-constrained multi-DAGs based on reverse HEFT," *2016 International Conference on Information System and Artificial Intelligence (ISAI)*, IEEE, 2016, pp. 196–202.

[58] Wu, P. and Ryu, M., "Best Speed Fit EDF Scheduling for Performance Asymmetric Multiprocessors," *Mathematical Problems in Engineering*, Vol. 2017, 2017.

[59] Xie, G., Zeng, G., Li, Z., Li, R., and Li, K., "Adaptive dynamic scheduling on multi-functional mixed-criticality automotive cyber-physical systems," *IEEE Transactions on Vehicular Technology*, Vol. 66, No. 8, 2017, pp. 6676–6692.

[60] Pendleton, S. D., Andersen, H., Du, X., Shen, X., Meghjani, M., Eng, Y. H., Rus, D., and Ang, M. H., "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, Vol. 5, No. 1, 2017, pp. 6.

[61] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., and et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 1–12.

[62] Lin, S.-C., Zhang, Y., Hsu, C.-H., Skach, M., Haque, M. E., Tang, L., and Mars, J., "The architectural implications of autonomous driving: Constraints and acceleration," *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 751–766.

[63] "Road vehicles — Functional safety." https://www.iso.org/standard/68383.html.

[64] 167, R. F. S., *Software considerations in airborne systems and equipment certification*, RTCA, Incorporated, 1992.

[65] McCausland, P., "Self-driving Uber car that hit and killed woman did not recognize that pedestrians jaywalk," *Retrieved January*, Vol. 29, 2019, pp. 2020.

[66] Vlasic, B. and Boudette, N. E., "Self-driving Tesla was involved in fatal crash, US says," *New York Times*, Vol. 302016, 2016.

[67] Mehmed, A., Antlanger, M., and Steiner, W., "The Monitor as Key Architecture Element for Safe Self-Driving Cars," *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, 2020, pp. 9–12.

[68] Jha, S., Banerjee, S., Tsai, T., Hari, S. K. S., Sullivan, M. B., Kalbarczyk, Z. T., Keckler, S. W., and Iyer, R. K., "ML-Based Fault Injection for Autonomous Vehicles: A Case for Bayesian Fault Injection," *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 112–124.

[69] Banerjee, S. S., Jha, S., Cyriac, J., Kalbarczyk, Z. T., and Iyer, R. K., "Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data," *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 586–597.

[70] Boroujerdian, B., Genc, H., Krishnan, S., Cui, W., Faust, A., and Reddi, V., "MAVBench: Micro aerial vehicle benchmarking," *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2018, pp. 894–907.

[71] "NVIDIA Orin SoC," https://nvidianews.nvidia.com/news/nvidia-introduces-drive-agx-orin-advanced-software-defined-platform-for-autonomous-machines.

[72] "Mobileye: The Evolution of EyeQ," https://www.mobileye.com/our-technology/evolution-eyeq-chip.

[73] Zhang, X., Lok, M., Tong, T., Lee, S. K., Reagen, B., Chaput, S., Duhamel, P. J., Wood, R. J., Brooks, D., and Wei, G., "A Fully Integrated Battery-Powered System-on-Chip in 40-nm CMOS for Closed-Loop Control of Insect-Scale Pico-Aerial Vehicle," *IEEE Journal of Solid-State Circuits*, Vol. 52, No. 9, 2017, pp. 2374–2387.

[74] Hashim, A., Saini, T., Bhardwaj, H., Jothi, A., and Kumar, A. V., "Application of Swarm Intelligence in Autonomous Cars for Obstacle Avoidance," *Integrated Intelligent Computing, Communication and Security*, Springer, 2019, pp. 393–404.

[75] Vega, A., Buyuktosunoglu, A., and Bose, P., "Towards "Smarter" Vehicles Through Cloud-Backed Swarm Cognition," *2018 IEEE Intelligent Vehicles Symposium*, IV 2018, 2018, pp. 1079–1086.

[76] "Coefficient of variation — Wikipedia, The Free Encyclopedia," https://en.wikipedia.org/w/index.php?title=Coefficient_of_variation.

[77] Chen, Y.-H., Krishna, T., Emer, J. S., and Sze, V., "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, Vol. 52, No. 1, 2016, pp. 127–138.

[78] Seok, M., Jeon, D., Chakrabarti, C., Blaauw, D., and Sylvester, D., "A 0.27 V 30MHz 17.7 nJ/transform 1024-pt complex FFT core with super-pipelining," *2011 IEEE International Solid-State Circuits Conference*, IEEE, 2011, pp. 342–344.

[79] "Mini-ERA: Simplified Version of the Main ERA Workload," https://github.com/IBM/mini-era.

[80] Frigo, M. and Johnson, S. G., "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, Vol. 93, No. 2, 2005, pp. 216–231, Special issue on "Program Generation, Optimization, and Platform Adaptation".

[81] "The ARM Computer Vision and Machine Learning library," https://github.com/ARM-software/ComputeLibrary.

[82] "Pyterbi: A PyCUDA and PP parallelized Viterbi decoder," https://github.com/loxodes/pyterbi.

[83] Redmon, J. and Farhadi, A., "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[84] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A., "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision*, Vol. 111, No. 1, Jan. 2015, pp. 98–136.

[85] Held, D., Thrun, S., and Savarese, S., "Learning to track at 100 fps with deep regression networks," *European Conference on Computer Vision*, Springer, 2016, pp. 749–765.

[86] Smeulders, A. W., Chu, D. M., Cucchiara, R., Calderara, S., Dehghan, A., and Shah, M., "Visual tracking: An experimental survey," *IEEE transactions on pattern analysis and machine intelligence*, Vol. 36, No. 7, 2013, pp. 1442–1468.

[87] Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D., "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE transactions on robotics*, Vol. 31, No. 5, 2015, pp. 1147–1163.

[88] Geiger, A., Lenz, P., Stiller, C., and Urtasun, R., "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, Vol. 32, No. 11, 2013, pp. 1231–1237.

[89] "ORB-SLAM2-GPU," https://github.com/yunchih/ORB-SLAM2-GPU2016-final.

[90] Darweesh, H., Takeuchi, E., Takeda, K., Ninomiya, Y., Sujiwo, A., Morales, L. Y., Akai, N., Tomizawa, T., and Kato, S., "Open source integrated planner for autonomous navigation in highly dynamic environments," *Journal of Robotics and Mechatronics*, Vol. 29, No. 4, 2017, pp. 668–684.

[91] Kato, S., Takeuchi, E., Ishiguro, Y., Ninomiya, Y., Takeda, K., and Hamada, T., "An open approach to autonomous vehicles," *IEEE Micro*, Vol. 35, No. 6, 2015, pp. 60–68.

[92] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W., "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees," *Autonomous Robots*, 2013, Software available at http://octomap.github.com.

[93] Hermann, A., Drews, F., Bauer, J., Klemm, S., Roennau, A., and Dillmann, R., "Unified GPU voxel collision detection for mobile manipulation planning," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 4154–4160.

[94] Devaurs, D., Siméon, T., and Cortés, J., "Parallelizing RRT on distributed-memory architectures," *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 2261–2266.

[95] Sucan, I. A., Moll, M., and Kavraki, L. E., "The open motion planning library," *IEEE Robotics & Automation Magazine*, Vol. 19, No. 4, 2012, pp. 72–82.

[96] Park, C., Pan, J., and Manocha, D., "RealTime GPU-based motion planning for task executions," *IEEE International Conference on Robotics and Automation Workshop on Combining Task and Motion Planning (May 2013)*, Citeseer, 2013.

[97] Umari, H. and Mukhopadhyay, S., "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 1396–1402.

[98] ARM, A., "AMBA AXI and ACE Protocol Specification," 2010.

[99] Binkert, N. L., Dreslinski, R. G., Hsu, L. R., Lim, K. T., Saidi, A. G., and Reinhardt, S. K., "The M5 simulator: Modeling networked systems," *Ieee micro*, Vol. 26, No. 4, 2006, pp. 52–60.

[100] Pillai, P. and Shin, K. G., "Real-time dynamic voltage scaling for low-power embedded operating systems," *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001, pp. 89–102.

[101] Aydin, H., Melhem, R., Mossé, D., and Mejía-Alvarez, P., "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, Vol. 53, No. 5, 2004, pp. 584–600.

[102] Flynn, M. and Luk, W., *Chip Basics: Time, Area, Power, Reliability, and Configurability*, 06 2011, pp. 39–73.

[103] Vega, A., Amarnath, A., Wellman, J.-D., Kassa, H., Pal, S., Franke, H., Buyuktosunoglu, A., Dreslinski, R., and Bose, P., "STOMP: A Tool for Evaluation of Scheduling Policies in Heterogeneous Multi-Processors," *arXiv preprint arXiv:2007.14371*, 2020.

[104] Garey, M. R. and Graham, R. L., "Bounds for multiprocessor scheduling with resource constraints," *SIAM Journal on Computing*, Vol. 4, No. 2, 1975, pp. 187–200.

[105] Davis, R. I. and Burns, A., "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems," *ACM Comput. Surv.*, Vol. 43, No. 4, Oct. 2011.

[106] Chronaki, K., Rico, A., Badia, R. M., Ayguadé, E., Labarta, J., and Valero, M., "Criticality-aware dynamic task scheduling for heterogeneous architectures," *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 329–338.

[107] Tong, Z., Deng, X., Chen, H., Mei, J., and Liu, H., "QL-HEFT: a novel machine learning scheduling scheme base on cloud computing environment," *Neural Computing and Applications*, 03 2019, pp. 1–18.

[108] Shetti, K. R., Fahmy, S. A., and Bretschneider, T., "Optimization of the HEFT Algorithm for a CPU-GPU Environment," *2013 International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2013, pp. 212–218.

[109] Capota, E. A., Stangaciu, C. S., Micea, M. V., and Curiac, D.-I., "Towards Mixed Criticality Task Scheduling in Cyber Physical Systems: Challenges and Perspectives," *Journal of Systems and Software*, 2019.

[110] Swaminathan, K., Kultursay, E., Saripalli, V., Narayanan, V., Kandemir, M. T., and Datta, S., "Steep-slope devices: From dark to dim silicon," *IEEE Micro*, Vol. 33, No. 5, 2013, pp. 50–59.

[111] Swaminathan, K., Liu, H., Sampson, J., and Narayanan, V., "An examination of the architecture and system-level tradeoffs of employing steep slope devices in 3D CMPs," *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, IEEE, 2014, pp. 241–252.

[112] Gopireddy, B., Skarlatos, D., Zhu, W., and Torrellas, J., "HetCore: TFET-CMOS hetero-device architecture for CPUs and GPUs," *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2018, pp. 802–815.

[113] Aly, M. M. S., Gao, M., Hills, G., Lee, C.-S., Pitner, G., Shulaker, M. M., Wu, T. F., Asheghi, M., Bokor, J., Franchetti, F., et al., "Energy-efficient abundant-data computing: The N3XT 1,000 x," *Computer*, Vol. 48, No. 12, 2015, pp. 24–33.

[114] Reagen, B., Adolf, R., Shao, Y. S., Wei, G.-Y., and Brooks, D., "Machsuite: Benchmarks for accelerator design and customized architectures," *2014 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, 2014, pp. 110–119.

[115] Shao, Y. S., Xi, S. L., Srinivasan, V., Wei, G.-Y., and Brooks, D., "Co-designing accelerators and soc interfaces using gem5-aladdin," *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–12.

[116] "STOMP," https://github.com/IBM/stomp.

[117] Vega, A., Wellman, J.-D., Franke, H., Buyuktosunoglu, A., Bose, P., Amarnath, A., Kassa, H., Pal, S., and Dreslinski, R., "STOMP: Agile Evaluation of Scheduling Policies in Heterogeneous Multi-Processors," *DOSSA-3 Workshop@ HPCA*, 2021.