Energy-Efficient Computing with Fine-Grained Many-Core Systems

By

BIN LIU
B.S. (Shanghai Jiao Tong University, Shanghai, China), 2007
M.S. (University of California, Davis), 2010

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Dr. Bevan M. Baas, Chair

Dr. Soheil Ghiasi

Dr. Rajeevan Amirtharajah

Committee in charge
2016

Abstract

For the past half century, Moore's Law has been the fundamental driver of high-performance computing. The continued CMOS technology scaling doubles the transistor density of VLSI systems and had provided a predictable 40% performance improvement of single-core processors for every 18 to 24 months. However, as Dennard Scaling ends, the era of scaling frequency and performance without increasing power density is over. Since 2005, the semiconductor industry shifted to multi-core and many-core processors in order to sustain the proportional scaling of performance along with transistor count increases. One of the critical challenges for many-core system design is to reduce the power dissipation and improve the energy efficiency of the chip. Researchers are eager to seek innovative low power architectures and techniques to relieve the "dark silicon" problem and effectively convert transistors to performance.

To demonstrate that many-core processors with network-on-chip interconnects is a promising architecture for high-performance energy-efficient computing, 16 Advanced Encryption Standard (AES) engines are proposed on a fine-grained many-core system by exploring different granularities of data-level and task-level parallelism. The smallest design utilizes only six cores for offline key expansion and eight cores for online key expansion, while the largest requires 107 cores and 137 cores, respectively. In comparison with published AES cipher implementations on general purpose processors, the designs have has 3.5–15.6 times higher throughput per unit of chip area and 8.2–18.1 times higher energy efficiency. Moreover, the design shows 2.0 times higher throughput than the TI DSP C6201, and 3.3 times higher throughput per unit of chip area and 2.9 times higher energy efficiency than the GeForce 8800 GTX.

Next, a scalable joint local and global dynamic voltage and frequency scaling (D-VFS) scheme is proposed to further improve the energy efficiency for many-core systems by monitoring on-line workload variations. The local algorithms selects the voltage and frequency pair for each individual core based on its FIFO occupancy and stall information, while

the global algorithm tunes the global voltage supplies based on the workload of all active processors. To demonstrate the effectiveness of the proposed solution, a suite of benchmarks are tested on a many-core globally asynchronous locally synchronous (GALS) platform. The experiment results show that the proposed approach can achieve near-optimal power saving under performance constraints. Different local algorithms are compared in terms of power saving, voltage switching frequency and response delay to workload variation. The impact of the number of voltage supplies and global voltage tuning resolution on the global algorithm is also investigated.

To further improve the energy efficiency beyond traditional DVFS, core scaling is proposed by introducing an extra dimension beyond supply voltage and clock frequency scaling. This dissertation addresses the problem of minimizing the power dissipation of many-core systems under performance constraints by choosing an appropriate number of active cores and per-core voltage/frequency levels. A genetic algorithm based solution is proposed to solve the problem. Experiments with real applications show that (1) dynamically scaling the number of active cores can improve the energy efficiency by 5% to 42% compared with per-core DVFS for different performance requirements; (2) core scaling favors systems with more global voltage supplies and high-performance leaky process when the performance requirement is loose, while it favors systems with fewer global voltage supplies and low-power less-leaky process when the performance requirement is tight; (3) increasing the number of global voltage supplies or leakage ratio can reduce the optimal core count by 22% and 50%, respectively.

To my wife Yun, my daughter Hannah, and my parents.

Acknowledgments

When I started to write this page, my journey of pursuing the PhD degree is coming to the end. I have never realized how difficult a task it would be to say "thank you" for all the help, support and encouragement given by so many people along the way. Simply and sincerely, I could not have done it on my own, and there can be no overstating the contributions of a large number of people who have made the experience rewarding, wonderful and unforgettable.

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Bevan Baas, for his valuable guidance, consistent encouragement and remarkable patience throughout this research. I am grateful that he always made himself available for discussion and advice despite his busy schedule. His insights, wisdom, enthusiasm and dedication have not only been a great resource of inspiration for this work, but also taught me how to become a good and respectful researcher, and would substantially influence my future career.

I would like to thank Professor Soheil Ghiasi and Professor Rajeevan Amirtharajah for serving on my dissertation committee and providing invaluable feedback on my research. I would also like to thank Professor Hussain Al-Asaad and Professor Charles Martel for serving on my qualification committee and giving insightful advice on my research proposal. I would like to thank Professor Shu Lin, Professor Kent Wilken, Professor Venkatesh Akella, Professor Soheil Ghiasi, Professor Rajeevan Amirtharajah and Professor Zhi Ding for their exceptional lectures that help me build solid foundations for my research work. I would also like to thanks all of the staff of the Electrical and Computer Engineering Department at UC Davis for all of their support.

Being a member of the VLSI Computation Laboratory is a privilege since I had the opportunity to meet and work with a group of smart and fun people. I would like to thank my friend Dean Truong for his endless help and enlightening discussions. I would like to thank the fellows who worked on KiloCore tape out together: Jon Pimentel, Brent Bohnenstiehl, Aaron Stillmaker, and Emmanuel Adeagbo. I will never forget the days and nights we put together to make the miracle. I would also like to thank all the previous and current VCL members: Tinoosh Mohsenin, Anh Tran, Zhibin Xiao, Stephen Le, Lucas Stillmaker, Houshmand Mehr, Nima Mostafavi, Trevin Murakami, Jeremy Webb, Shifu Wu, Michael Braly, Timothy Andreas, and Satyabrata Sarangi. I am grateful to have a chance to work with all of these colleagues at UC Davis.

I would like to express my special appreciation to my beloved wife Yun Duan. This dissertation would not be possible without her endless love and support. She has been the one to encourage me when I was stressed out, and the one to share my accomplishments and successes. Thanks for being my best friend and my best companion. I would also like to thank my parents, my relatives and my lovely daughter Hannah, for all of their love and support.

Finally, I gratefully acknowledge support from US National Science Foundation (NSF) CCF Grants 1018972 and 0903549, NSF CAREER Award 0546907, SRC GRC Grants 1598, 1971 and 2321, CSR Grant 1659, C2S2 Grant 2047.002.014, UC Micro, Intel, Intellasys, UC Davis Summer Researcher Award and UC Davis Travel Grant. I also would like to thank ST Microelectronics for donating the chip fabrication.

Contents

Abstract **ii**

Acknowledgments **v**

List of Figures **x**

List of Tables **xii**

1 Introduction **1**
1.1 From Single To Many . . . 1
1.2 Dissertation Contributions and Outline . . . 4

2 Background **8**
2.1 Power and Energy Consumption in CMOS Circuits . . . 8
2.1.1 Dynamic Power . . . 8
2.1.2 Short-Circuit Power . . . 9
2.1.3 Leakage Power . . . 9
2.2 Low Power Techniques . . . 11
2.3 Targeted Many-Core Architecture . . . 14
2.3.1 Fine-grained Many-core Architecture . . . 14
2.3.2 Asynchronous Array of Simple Processors . . . 15
2.3.3 Programming Methodology on AsAP . . . 17

3 Parallel AES Engines for Many-Core Processor Arrays **20**
3.1 Introduction . . . 20
3.2 Advanced Encryption Standard . . . 22
3.3 AES Implementations on AsAP . . . 24
3.3.1 One-task One-processor . . . 25
3.3.2 Loop-unrolled Nine Times . . . 26
3.3.3 Loop-unrolled Three Times . . . 26
3.3.4 Parallel-MixColumns . . . 29
3.3.5 Parallel-SubBytes-MixColumns . . . 29
3.3.6 Full-parallelism . . . 32
3.3.7 Small . . . 32
3.3.8 No-merge-parallelism . . . 34
3.3.9 Designs with Longer Keys . . . 35
3.4 Area Efficiency Analysis . . . 35

3.4.1 Area Optimization Methodology . . . 36
3.4.2 Area Efficiency Comparison . . . 36
3.5 Energy Efficiency Analysis . . . 41
3.5.1 Power Numbers from Chip Measurements . . . 41
3.5.2 Power Estimation Methodology . . . 43
3.5.3 Energy Efficiency Comparison . . . 45
3.6 Related Work and Comparison . . . 48
3.7 Conclusion . . . 51

4 Scalable Joint Local and Global Dynamic Voltage and Frequency Scaling for Many-Core Systems 54
4.1 Introduction . . . 54
4.1.1 Related Work . . . 55
4.1.2 Chapter Organization and Contributions . . . 57
4.2 Preliminaries . . . 57
4.2.1 Power Model . . . 57
4.3 Frequency Scaling Versus Voltage Dithering . . . 59
4.4 Proposed DVFS Local Algorithm . . . 66
4.4.1 Performance Constrained Systems . . . 66
4.4.2 Dithering with FIFO Occupancy . . . 66
4.4.3 Dithering with FIFO Stall Information . . . 73
4.5 Proposed DVFS Global Algorithm . . . 76
4.6 Experimental Results . . . 79
4.6.1 Benchmarks . . . 80
4.6.2 Case Study: 9-core AES Engine . . . 83
4.6.3 Local Algorithm Comparison . . . 84
4.6.4 Global Algorithm Evaluation . . . 87
4.7 Conclusion . . . 91

5 Optimizing Power of Many-Core Systems by Exploiting Dynamic Voltage, Frequency and Core Scaling 92
5.1 Introduction . . . 92
5.2 Problem Formulation . . . 95
5.3 Proposed Algorithm . . . 98
5.4 Methodology . . . 102
5.4.1 Power Model . . . 102
5.4.2 Benchmark – Eight AES Engines . . . 105
5.5 Experimental Results . . . 105
5.5.1 Per-Core DVFCS VS. Per-Core DVFS . . . 105
5.5.2 Different Number of Global Voltage Supplies . . . 110
5.5.3 Different Leakage Ratio . . . 114
5.6 Conclusion . . . 118

6 Conclusion and Future Work 120
6.1 Conclusion . . . 120
6.2 Future Work . . . 121

Glossary 123

Related Publications **126**

Bibliography **128**

List of Figures

1.1 The maximum clock frequency trend of processors . . . 2
1.2 The trend of the number of cores on single die . . . 3
1.3 Die photos of four many-core processor arrays . . . 5

2.1 Leakage power trends in total power consumption . . . 10
2.2 Illustration of energy saving with voltage and frequency scaling . . . 13
2.3 Block diagram of AsAP2 . . . 16
2.4 Programming methodology on fine-grained many-core processors. . . . 18

3.1 Block diagram of AES encryption. . . . 23
3.2 Dataflow and AsAP mapping of the One-task One-processor AES engine . . . 26
3.3 Dataflow and AsAP mapping of the Loop-unrolled Nine Times AES engine . . . 27
3.4 Dataflow and AsAP mapping of the Loop-unrolled Three Times AES engine . . . 28
3.5 Dataflow and AsAP mapping of the Parallel-MixColums AES engine . . . 30
3.6 Dataflow and AsAP mapping of the Parallel-SubBytes-MixColumns AES engine . . . 31
3.7 Dataflow and AsAP mapping of the Full-parallelism AES engine . . . 33
3.8 AsAP mapping of the Small AES engine . . . 34
3.9 AsAP mapping of the No-merge-parallelism AES engine . . . 34
3.10 Dataflow and AsAP mapping of the optimized Full-parallelism AES engine . . . 37
3.11 Throughput versus the number of cores for offline AES engines . . . 40
3.12 Maximum frequency and power dissipation of one AsAP2 core . . . 42
3.13 Energy efficiency for AES engines . . . 46
3.14 Energy overhead for online AES engines . . . 47
3.15 Delay and power scaling model for different technology nodes . . . 49
3.16 Area and energy efficiency for different software AES platforms . . . 52

4.1 Workload variation on many-core systems . . . 55
4.2 Power model of the targeted many-core system . . . 58
4.3 Energy efficiency of different voltage and frequency scaling methods . . . 60
4.4 Voltage noise during voltage switches on AsAP . . . 61
4.5 Comparison between frequency scaling and voltage dithering . . . 62
4.6 Break even number of clock cycles for $E_{dith} < E_{freq}$ with different LR . . . 63
4.7 Break even number of clock cycles for $E_{dith} < E_{freq}$ with different V_{high} . . . 64
4.8 Upstream and downstream performance constraints . . . 65
4.9 Example of FIFO partitions . . . 68
4.10 9-core AES engine dataflow diagram. . . . 73
4.11 Example of workload distribution change . . . 78

4.12 Six basic dataflow patterns . . . 81
4.13 Power saving of individual cores in the 9-core AES engine. . . . 84
4.14 Power saving of the proposed local algorithms for benchmarks in Table 4.1. . . . 85
4.15 Comparison of local DVFS algorithms . . . 86
4.16 Power saving comparison between DVFS with and without global optimization . . . 88

5.1 Illustration of core scaling problem . . . 97
5.2 Chromosome block diagram and solution example of the proposed GA . . . 98
5.3 Crossover and mutation in GA . . . 101
5.4 Frequency, dynamic power and normalized leakage factor . . . 103
5.5 AsAP mapping of the 59-core AES engine . . . 104
5.6 Power saving of different voltage/frequency/core scaling methods . . . 106
5.7 Dynamic and leakage power percentage for the 59-core AES engine . . . 108
5.8 Dynamic and leakage power saving for DVFS and DVFCS versus throughput . . . 109
5.9 Optimal core count selected by core scaling with different number of V_{DD}s . . . 112
5.10 Extra power saving brought by core scaling with different number of V_{DD}s . . . 113
5.11 Optimal core count selected by core scaling for different leakage ratios . . . 115
5.12 Extra power saving brought by core scaling for different leakage ratios . . . 116

List of Tables

3.1 Execution delays of AES blocks on AsAP2 25
3.2 Throughput and the number of cores for different AES engines 39
3.3 Active, stall and leakage power for one AsAP core 42
3.4 Execution cycles and power for offline Full-parallelism AES engine 44
3.5 Comparison of AES engines on different software platforms 50
3.6 Summary of state-of-the-art ASIC and FPGA AES implementations 53

4.1 Number of cores and throughput of different benchmarks 82
4.2 Comparison of optimal frequency and DVFS-selected frequency 83
4.3 Summary of Local Algorithms Performance Evaluation 87
4.4 Power saving of DVFS with and without global optimization 90
4.5 Power saving from global optimization with different tuning resolutions 91

5.1 Comparison of Various Low Power Design Techniques 93
5.2 Number of cores and throughput of different AES engines 104
5.3 Power saving from DVFS and DVFCS with different number of V_{DD}s 111
5.4 Power saving from DVFS and DVFCS for different leakage ratios 117

Chapter 1

Introduction

1.1 From Single To Many

For the past half century, Moore's Law [1] has been the fundamental driver for semiconductor industry, as it predicted that the continued CMOS technology improvement doubles the available number of transistors on the same die area for every 18 to 24 months. Historically, Dennard Scaling was the primary force that supported Moore's Law. According to Dennard Scaling [2], if the lithographic dimensions and supply voltages of CMOS are scaled simultaneously by a factor of $\kappa = \sqrt{2}$, the power density keeps constant even though the density of transistors is doubled and the switch speed of transistors is 40% faster. The microprocessor architects exploited the smaller, faster and more power efficient transistors provided by CMOS technology scaling, and transferred the extra transistors into new techniques, such as pipelining, branch prediction, out-of-order execution, and large caches, to further improve the performance of single-core processors. By combining the speed up from devices and the advanced architecture innovations, the single-core processor performance sustained a 52% annual improvement rate between 1986 to 2002 [3].

However, since early 2000s, Dennard Scaling started to fail and the supply voltage stopped scaling down along with the transistor dimensions. There are three major reasons behind that. First of all, as the CMOS technology scaling, the leakage power becomes a substantial portion of the power consumption. The leakage current increases exponentially when the threshold voltage is decreased. To keep the subthreshold leakage power under control, the threshold voltage could not be lowered further [4]. Secondly, because of the growth in gate oxide tunneling current, gate

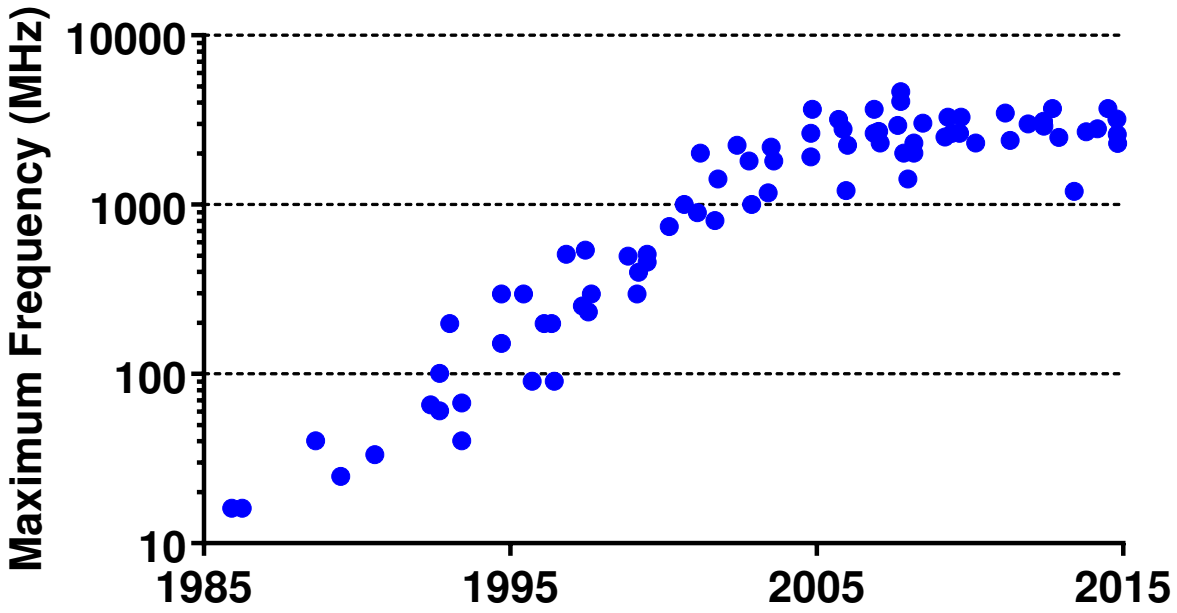


Figure 1.1: The maximum clock frequency trend of processors [8]. Major processors reported by industry manufacturers.

oxide scaling was also slowed down due to leakage power constraints [5]. Finally, as transistors approached atomic dimensions, the process variation problem became more prominent that also limited the supply voltage scaling [6]. Overall, the supply voltage has been stayed at around 1 V since 90 nm [7].

The failure of Dennard Scaling leads to per-transistor switching power not scaling down at the pace of Moore's Law, and causing the power density to increase with successive generations. The surface temperature of high performance processors were expected to reach as high as a nuclear reactor by 2005, a rocket nozzle by 2010 and the surface of the sun by 2015, if the frequency had kept scaling as the existing trajectory [9]. This is recognized as "*Power Wall*". Therefore, the maximum frequency of processors stopped increasing since around 2005 as shown in Fig. 1.1. Additionally, the growing disparity of speed between CPU and memory ("*Memory Wall*"), and the lack of instruction level parallelism (ILP) due to the essential complexity in modern programs ("*ILP Wall*") diminish the performance improvement from building more aggressive pipelining and complex superscalar single-core processors. Consequently, the computing industry shifted to the era of multi-core processors in order to sustain the performance improvement at a historical rate

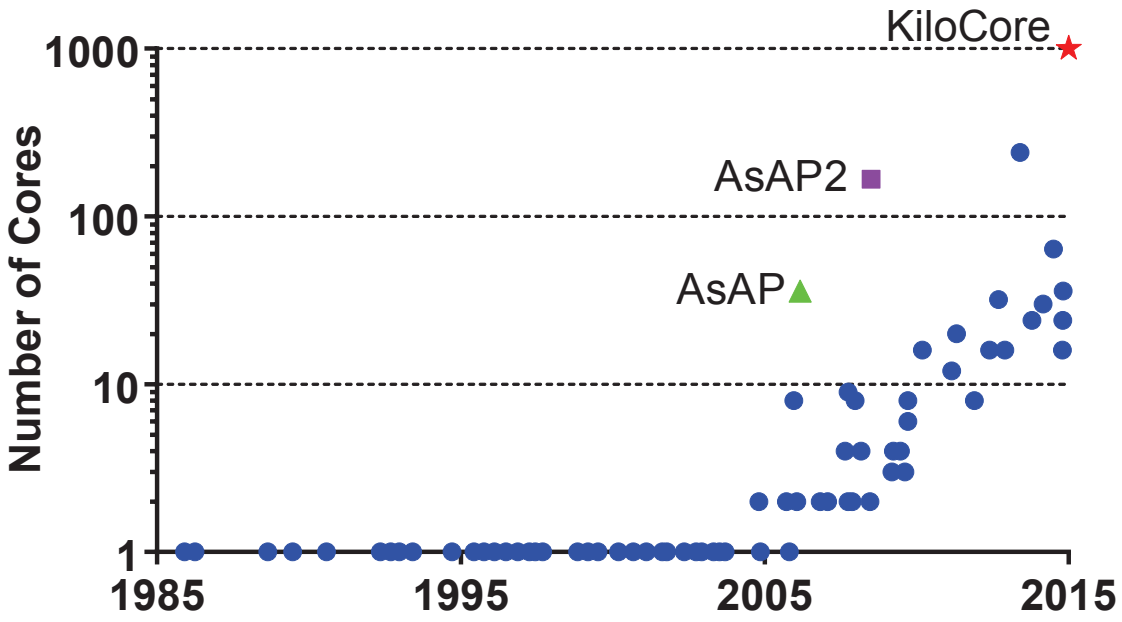


Figure 1.2: The trend of the number of cores on single die [18]. Major processors reported by both industry manufacturers and university research labs.

along with transistor count increase by exploring thread-level, task-level and data-level parallelism in addition to ILP. Since 2005, all major processor manufactures, such as Intel, AMD, Sun and IBM, started to deliver their dual-core processors [10, 11, 12, 13]. Later, the number of cores kept scaling as the CMOS technology. As shown in Fig. 1.2, quad-core, hexa-core and even octo-core processors were released to the market [14, 15, 16, 17].

However, after only half decades of the first introduction of multi-core processors, the era of simply increasing the number of cores came to its end. Due to the lack of high degree of parallelism and severe energy inefficiency in the transistor level, adding more cores would not be able to improve the performance at the historic rate [19]. It is predicted that without introducing novel low power architectures or techniques, more than 50% of the chip has to be turned off due to power concerns when CMOS technology shrinks to 8 nm [20]. The lack of performance benefits and the lack of ability to utilize all the transistors provided by new process technologies bring a gap between the projected performance of multi-core processors and historical anticipation that is as large as 24× [21]. Researchers and computing industry are eager to seek innovative low power architectures and techniques to relieve the "dark silicon" problem and effectively convert transistors

to performance.

Recently, instead of integrating more complex cores on the same die, both industry and academia are actively exploring the design space of many-core architectures. *Fine-grained multiple instruction multiple data (MIMD) many-core processor array with network-on-chip interconnects* has been demonstrated as promising architectures for high-performance energy-efficient computing. Examples include the Tilera 64-core TILE64 [22] and 72-core TILE-Gx72 [23], Intel 80-core TeraFLOPs processor [24] and 48-core IA-32 processor [25], Kalray 256-core many-core processor array [26], UC Davis 36-core AsAP [27], 167-core AsAP2 [28] and 1000-core KiloCore [29]. Fig. 1.3 shows the die photos of four of the above many-core processor arrays. In the fine-grained many-core processor array, each core is smaller, simpler and delivers lower performance than the traditional large complex cores. However, the total computation throughput is much higher by inherently exploring parallelism from application-level, task-level, and data-level. Additionally, each core can be voltage and frequency scaled individually that results in fine grain power management to improve the energy efficiency. Fine-grained many-core processor array has shown its advantage in terms of energy efficiency compared with traditional processors in various of applications, such as wireless baseband [30, 31, 32], video processing [33, 34, 35, 36], biomedical signal processing [37], encryption [38, 39, 40], sorting [41], and error control coding [42].

1.2 Dissertation Contributions and Outline

As the number of cores on a single die keep increasing, several challenges are required to be addressed such as homogeneous and heterogeneous design, power and thermal management, network-on-chip interconnection, memory architecture, programming models, application mapping, and task scheduling. This dissertation focuses on designing and implementing new low power techniques to improve the energy efficiency of fine-grained many-core processors under performance constraints. The major contributions are listed as follows.

- It presents 16 Advanced Encryption Standard (AES) engines on a fine-grained many-core processor array by exploring different granularities of data-level and task-level parallelism. The design space and the trade-off between performance and the number of cores are examined comprehensively. The smallest design utilizes only 6 cores for offline key expansion and 8 cores

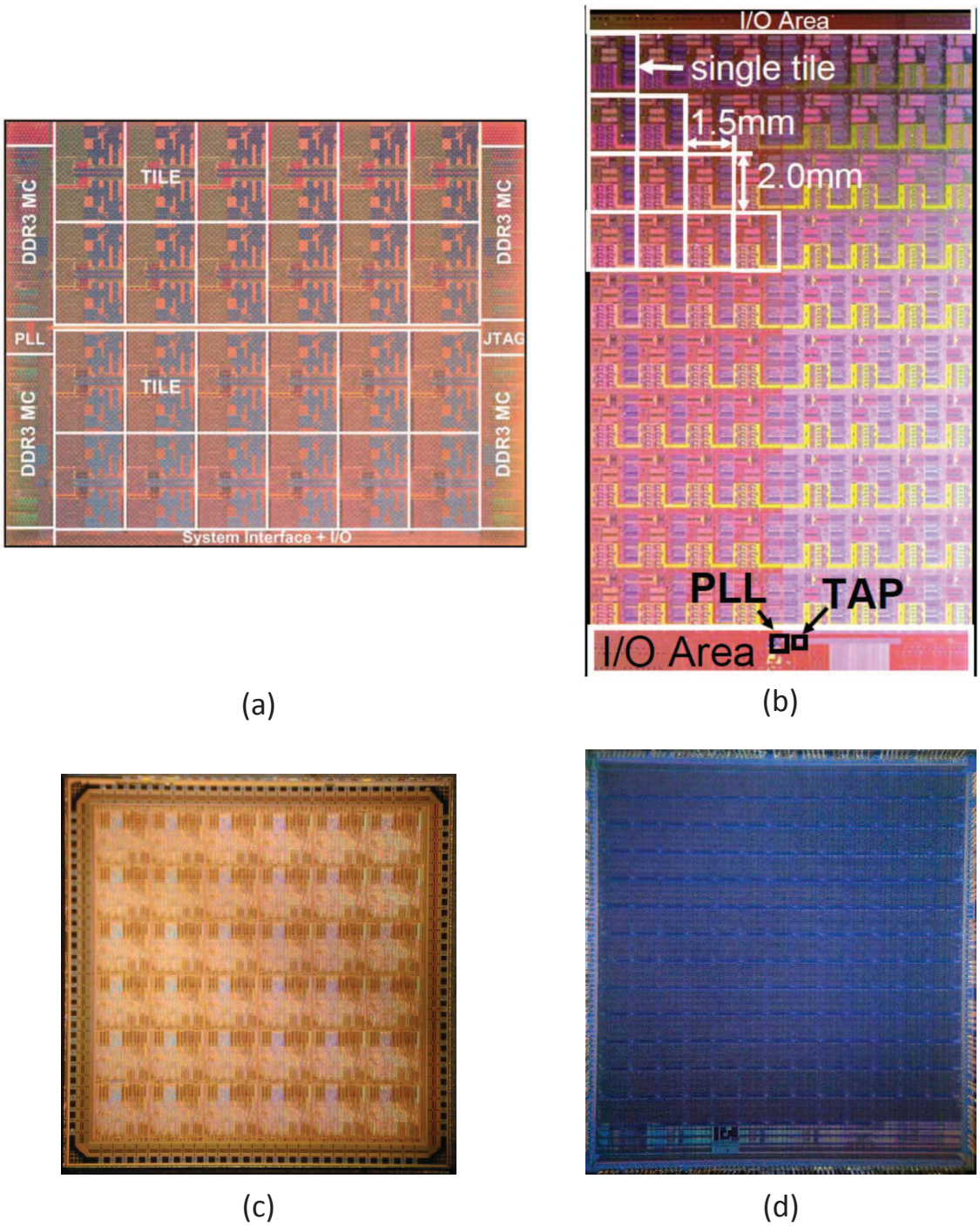


Figure 1.3: Some die photos of existing many-core processor arrays. (a) Intel 48-core; (b) Intel 80-core; (c) AsAP; (d) AsAP2.

for online key expansion, while the largest requires 107 cores and 137 cores, respectively. In comparison with published AES cipher implementations on general purpose processors, the designs have 3.5–15.6 times higher throughput per unit of chip area and 8.2–18.1 times higher energy efficiency. Moreover, the design shows 2.0 times higher throughput than the TI DSP C6201, and 3.3 times higher throughput per unit of chip area and 2.9 times higher energy efficiency than the GeForce 8800 GTX.

- It proposes an online scalable hardware-based joint local and global dynamic voltage and frequency scaling (DVFS) solution driven by workload variations for many-core processors. The local algorithms is used to select the voltage and frequency pair for each individual core based on its FIFO occupancy and stall information, while the global algorithm tunes the global voltage supplies based on the workload of all active cores. To demonstrate the effectiveness of the proposed solution, a suite of benchmarks are tested on a many-core globally asynchronous locally synchronous (GALS) platform. The experiment results show that the proposed approach can achieve near-optimal power saving under performance constraints. Different algorithms are compared in terms of power saving, voltage switching frequency and the response delay to workload variation. The impact of the number of voltage supplies and the tuning resolution of voltage regulators on the global optimization are also investigated.

- It addresses the problem of minimizing the power dissipation of many-core systems under performance constraints by choosing an appropriate number of active cores and per-core voltage/frequency levels (DVFCS). A genetic algorithm based solution is proposed to solve the problem. Experiments with real applications show that (1) dynamically scaling the number of active cores can improve the energy efficiency by 5% to 42% compared with per-core DVFS for different performance requirements; (2) core scaling favors systems with more global voltage supplies and high-performance leaky process when the performance requirement is loose, while it favors systems with fewer global voltage supplies and low-power less-leaky process when the performance requirement is tight; (3) increasing the number of global voltage supplies or leakage ratio can reduce the optimal core count by 22% and 50%, respectively.

The dissertation is organized as follows. Chapter 2 covers the background information, including power and energy dissipation in CMOS circuits, low power techniques and the advantages

of fine-grained many-core processor array. It also introduces the targeted many-core processors and describes the programming and mapping methodology. Chapter 3 presents the 16 AES cipher implementations in detail and provides thorough performance and energy efficiency analysis and comparison. Chapter 4 describes the proposed joint local and global DVFS algorithms and demonstrates their effectiveness with real-world application benchmarks. Chapter 5 presents the motivation and the algorithm for DVFCS. It also investigates the extra power saving brought by the core scaling compared with traditional DVFS for systems with different performance requirements, leakage power ratio and the number of global voltage supplies. Finally, Chapter 6 concludes the dissertation and discusses possible research directions in the future.

Chapter 2

Background

2.1 Power and Energy Consumption in CMOS Circuits

The power dissipation in CMOS circuits can be divided into three main components: dynamic power, short-circuit power and leakage power [43].

2.1.1 Dynamic Power

The dynamic power dissipation (P_{dyn}) is the power utilized to charge and discharge the load capacitance at the output of individual CMOS logic cells during state transitions. In a CMOS design, when the output state toggles from "0" to "1", the load capacitance C_L gets charged through the PMOS transistors from 0 to V_{DD} by consuming a certain amount of energy from the power supply. Part of the energy is dissipated as heat during the charging process, while the remainder is stored on the load capacitance. When the output state changes from "1" to "0", the stored energy in C_L is dissipated through NMOS devices. The total amount of energy dissipated for each charge and discharge cycle can be represented as

$$E_{V_{DD}} = \int_0^\infty i_{V_{DD}}(t) V_{DD} dt = C_L V_{DD} \int_0^{V_{DD}} dv_{out} = C_L V_{DD}^2 \tag{2.1}$$

Considering the circuit switching frequency is f, and the probability of output state transition is α, then the dynamic power consumption can be obtained as

$$P_{dyn} = \alpha C_L V_{DD}^2 f = C_{eff} V_{DD}^2 f \tag{2.2}$$

where α and C_L can be combined as C_{eff}, which is the effective switching capacitance.

2.1.2 Short-Circuit Power

When the input voltage level transits from one state to another (high to low, or low to high), the PMOS and NMOS networks are conducting simultaneously for a short period of time (t_{sc}) due to the finite slope of the input signal. As a result, a conducting path is formed between V_{DD} and ground for t_{sc} and leads to a short-circuit energy dissipation per switching as

$$E_{sc} = V_{DD} I_{sc} t_{sc} \tag{2.3}$$

Similar to the dynamic power, the average short-circuit power consumption is also proportional to the switching activity and can be obtained as

$$P_{sc} = \alpha V_{DD} I_{sc} t_{sc} f = C_{sc} V_{DD}^2 f \tag{2.4}$$

where C_{sc} is the effective capacitance for short-circuit, since $\alpha I_{sc} t_{sc}$ has units of electronic charge. The short-circuit power becomes less important and negligible in deep submicrometer CMOS, since supply voltages scales much faster than threshold voltages [44].

2.1.3 Leakage Power

Historically, dynamic power is the dominant component in the total power dissipation. However, as the CMOS technology scaling, leakage power becomes more and more significant. As shown in Fig. 2.1, the ratio of leakage power to total power consumption has increased to approximate 50% in 2006. The source of the leakage power includes reverse-bias pn junction leakage, subthreshold leakage, gate tunneling current, gate current due to hot-carrier injection, gate induced drain leakage, and channel punchthrough current [46]. Gate tunneling leakage and subthreshold leakage are the two primary sources of static power [47].

The increase of gate tunneling leakage is caused by the continued oxide thickness scaling. The high electric field coupled with low oxide thickness results in tunneling of electrons from gate to substrate and also from substrate to gate through the oxide.

Subthreshold leakage is caused by the incomplete shut off of transistors. A partially conducting state of transistors leads a drain-source current, even when the gate-source voltage is smaller than the threshold voltage. The subthreshold current can be approximated by the following

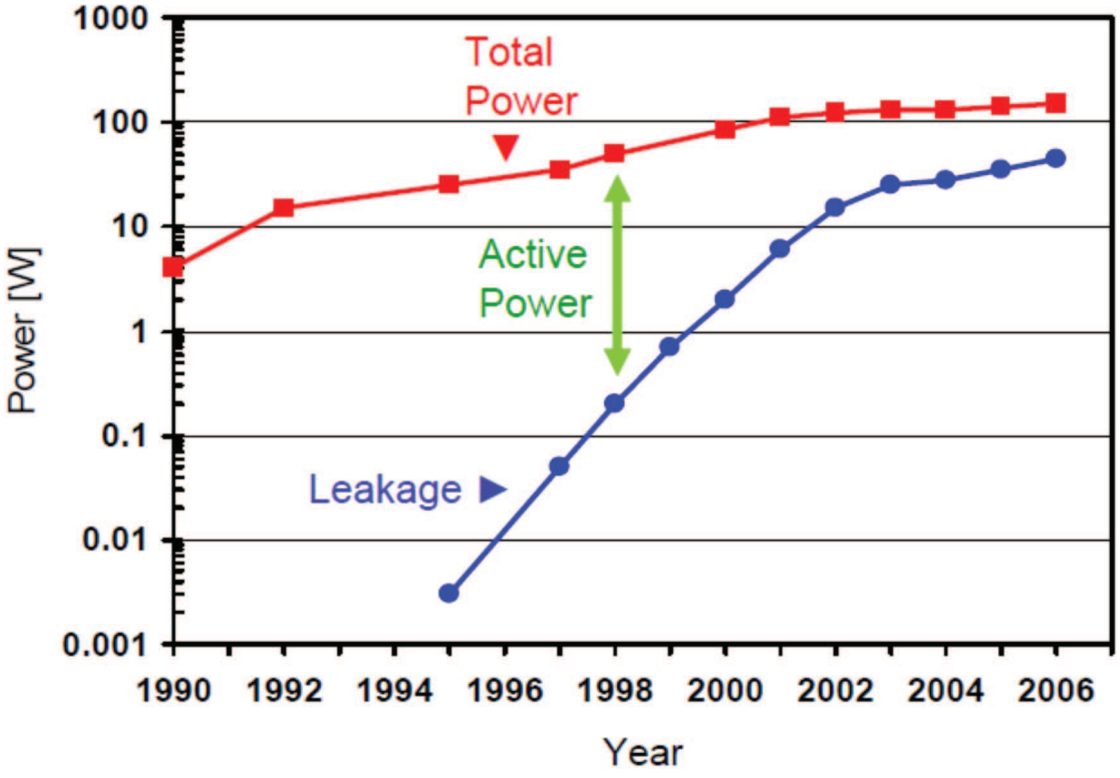


Figure 2.1: Leakage power trends in total power consumption [45].

equation [46],

$$I_{sub} = I_o \exp\left(\frac{V_{GS} - V_T}{nV_{th}}\right)\left(1 - \exp\left(-\frac{V_{DS}}{V_{th}}\right)\right) \tag{2.5}$$

where $I_o = \mu_0 C_{ox} \frac{W}{L} V_{th}^2 e^{1.8}$, V_T is the threshold voltage, $V_{th} = kT/q$ is the thermal voltage, C_{ox} is the gate oxide capacitance, and μ_0 is the zero bias mobility. Clearly, subthreshold leakage is a strong function of the threshold voltage V_T, since it appears in exponential terms. Historically, the threshold voltage was high enough that the subthreshold current is negligible. However, the subthreshold current is increased exponentially as the threshold voltage scaling down along with transistor lithographic dimensions. Therefore, even at $V_{GS} = 0\ V$, devices still leak. Considering the billions of transistors on the chip, leakage power becomes a significant portion of the total power consumption.

By combining all the components discussed above, the total power dissipation of CMOS circuits can be obtained as

$$\begin{aligned} P_{total} &= P_{dyn} + P_{sc} + P_{leak} \\ &= (\alpha C_L + C_{sc}) V_{DD}^2 f + I_{leak} V_{DD} \\ &= C_{eff} V_{DD}^2 f + I_{leak} V_{DD} \end{aligned} \tag{2.6}$$

The short-circuit power dissipation can be merged into dynamic power consumption since both of them happens during state transitions. As a result, the new C_{eff} is obtained by combining αC_L and C_{sc}.

2.2 Low Power Techniques

The most commonly used technique to reduce dynamic power is *clock gating* [48]. It selectively disables the clock inputs of registers that are not involved in carrying out useful computation, and reduces unnecessary switching activities. Additionally, it also reduces the load of the clock distribution network by disabling the inactive portions of the clock tree [49]. Overall, it decreases dynamic power dissipation linearly by reducing the overall effective load capacitance.

Power gating is a widely used technique to reduce leakage power. It uses sleep transistors to shut-off the idle function blocks from power supply. The sleep transistors are in series of pull-up and/or pull down networks, and usually built by devices with relative high V_{th} [50]. The

concept of power gating is simple, but the real design requires careful considerations. Increasing the size of sleep transistors more than necessary would add extra load capacitance, while sizing them too small would result in a supply current limitation and performance degradation [51]. Additionally, designed power gates can induce significant ground bounce effect that can reduce the noise margin and induce voltage fluctuations in the power distribution network [52]. The granularity of power gating can go as fine as a few standard cells [53], to as coarse as memory blocks and even processors [54].

Multi-V_{th} is another technique to reduce leakage power dissipation, by utilizing transistor libraries with multiple threshold voltages on the same die. Transistors with higher V_{th} have higher performance, but higher leakage power as well. On the other hand, transistors with lower V_{th} have a larger delay but less leakage power dissipation. The logic gates on non-critical paths can be assigned as high V_{th} during design time to reduce leakage power, as long as there is no performance penalty [55, 56]. The multi-V_{th} can be also combined with multi-V_{DD} [57, 58], and sizing [59] for further power optimization.

Instead of selecting V_{th}s statically, *adaptive body bias* is capable of tuning the V_{th}s of gates during runtime, by controlling the transistor body-source voltage. During standby, a reverse body bias is applied to increase V_{th}, thus reducing leakage power dissipation [60, 61]. Alternatively, a forward body bias increases speed while increasing leakage power. However, body bias effect is less effective as the CMOS technology scaling due to short channel effects [62]. Besides reducing leakage power, adaptive body bias can also used to alleviate the impact of process, voltage, and temperature (PVT) variations [63, 64, 65].

Dynamic voltage and frequency scaling (DVFS) is an effective method to reduce dynamic power consumption by selecting a trade-off point between clock speed and power based on performance constraints and workload variations. The delay of logic gates is a function of its supply voltage, threshold voltage and a technology dependent constant β [66]:

$$t_{pd} \propto \frac{V_{DD}}{(V_{DD} - V_{th})^{\beta}} \tag{2.7}$$

Since the clock frequency f is directly proportional to the gate delay, it can be obtained

$$f \propto \frac{(V_{DD} - V_{th})^{\beta}}{V_{DD}} \approx V_{DD} \tag{2.8}$$

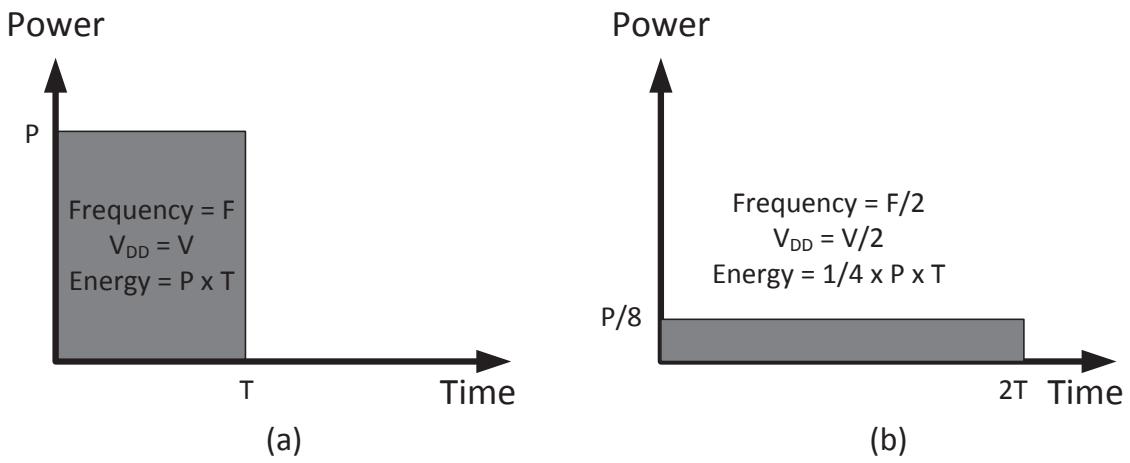


Figure 2.2: Illustration of dynamic energy saving with DVFS. (a) An application finishes in time T at frequency F and V. (b) Same application takes twice of the time at half frequency $F/2$ and half voltage $V/2$, while only consumes 1/4 of the initial energy.

where β is set to a typical value of 2. As shown in Eq. 2.6, dynamic power is proportional to $V_{DD}^2 \times f$. If clock frequency is scaling along with voltage supply as shown in Eq. 2.8, dynamic power decreases proportional to the cubic power of V_{DD}. Fig. 2.2 illustrates the power reduction and energy saving by scaling the voltage and frequency of a single task. As frequency F and voltage V, the application finishes in time T. If both the frequency and voltage are lowered to half of their initial value, the power is reduced to 1/8 of the initial value. Since the application takes twice of the time to finish, the total energy consumption is decreased to 1/4.

From the above example, the processor should run as slow as possible in order to achieve the largest power saving without violating any performance constraints. The most challenge part is to estimate the workload or slack execution time accurately. Various of methods have been discussed in the literature, including filtering [67, 68], dynamic slack reclamation [69, 70], FIFO and queue occupancy [71, 72, 73, 74], architectural runtime statistics [75, 76, 77]. In the above example, the power model is extremely simplified by only considering dynamic power. In reality, scaling down the voltage supply can also reduce the leakage power, but at a slower rate than the increase of execution time especially when V_{DD} is approaching to V_{th}. As a result, extensive voltage scaling might increase the total energy consumption when the extra leakage energy dissipation overwhelms the dynamic energy saving. Therefore, in some cases, it might be more energy efficient to finish computation faster than necessary and use power gating to cut off the leakage for the rest of the

time. Near-threshold computation has been proved to be the most energy efficient point for voltage scaling by balancing dynamic and leakage energy consumption, any further voltage scaling would impair the energy efficiency. [78, 79, 80, 81].

Recently, a combination of DVFS and scaling the number of active cores (DVFCS) has been proposed to further improve the energy efficiency and performance for many-core processors. Compared to DVFS, DVFCS is capable of tuning an extra dimension, the number of cores, besides voltage and frequency. The extra orthogonal dimension brings more flexibility for DVFCS to search the optimal energy efficiency solution with either per-chip level [82, 83] or per-core level DVFS [84].

2.3 Targeted Many-Core Architecture

2.3.1 Fine-grained Many-core Architecture

Performance

According to Pollack's Rule, the performance increase of an architecture is roughly proportional to the square root of its increase in complexity [85]. The rule implies that if the logic area is doubled in a processor, the performance of the core speeds up around 40%. On the other hand, a fine-grained many-core architecture has the potential to provide near linear performance improvement with complexity. For instance, instead of building a complicated core twice as large as before, a processor containing two cores (each is identical to the other) could achieve a possible 2× performance improvement if the application can be fully parallelized. Therefore, if the target application has enough inherent parallelism, an architecture with thousands of small cores would offer a better performance than one with a few large cores within the same die area [86].

Although the performance gain from many-core systems is highly dependent on the parallelism of the targeted applications according to Amdahl's Law. Task-level parallelism is naturally inherent and can be exploited easily in most of the DSP, embedded and even server applications, such as encryption [38], baseband communication [31], biomedical signal processing [37], video encoding [35], sorting [41] and MapReduce [87]. Moreover, multiple applications or multiple instances of one application can execute simultaneously on a single processor, which provides additional application-level parallelism so that the performance advantage of fine-grained many-core

architecture can be fully utilized.

Power

Besides performance improvement, fine-grained many-core architecture also provides several potential advantages on power issues.

- Each individual core can be individually voltage and frequency scaled based on its workload.
- Idle cores can be shut down by power gating, thereby reducing leakage.
- The power dissipation can be reduced by parallelizing or pipelining a serial task on a group of small cores [66].
- The workload can be balanced among different cores to avoid hot spots and produce a low die temperature. Dies with lower temperature have lower threshold voltage and less leakage [6].

2.3.2 Asynchronous Array of Simple Processors

The targeted Asynchronous Array of Simple Processors (AsAP) architecture is an example of a fine-grained many-core computation platform, supporting GALS on-chip network and per-core DVFS [88].

Fig. 2.3 shows the block diagram of AsAP2. The computational platform is composed of 164 small identical processors, three hardware accelerators and three 16 KB shared memories. All processors and shared memories are clocked by local fully independent oscillators and are connected by a reconfigurable 2D-mesh network that supports both nearby and long-distance communication [89]. Since each tile operates on its own frequency, the communication across different frequency domain is through dual-clock FIFOs [90]. Each tile on the platform can be statically configured to take input data from two links, while sending its output to other processors via dynamic configuration.

Each simple processor has a 6-stage pipeline, which issues one instruction per clock cycle. Moreover, no application-specific instructions are implemented. Each processor has a 128×32-bit instruction memory and a 128×16-bit data memory. Each processor occupies 0.17 mm^2 and has a maximum clock frequency of 1.2 GHz. The 167-processor chip was fabricated in 65 nm CMOS technology. [91].

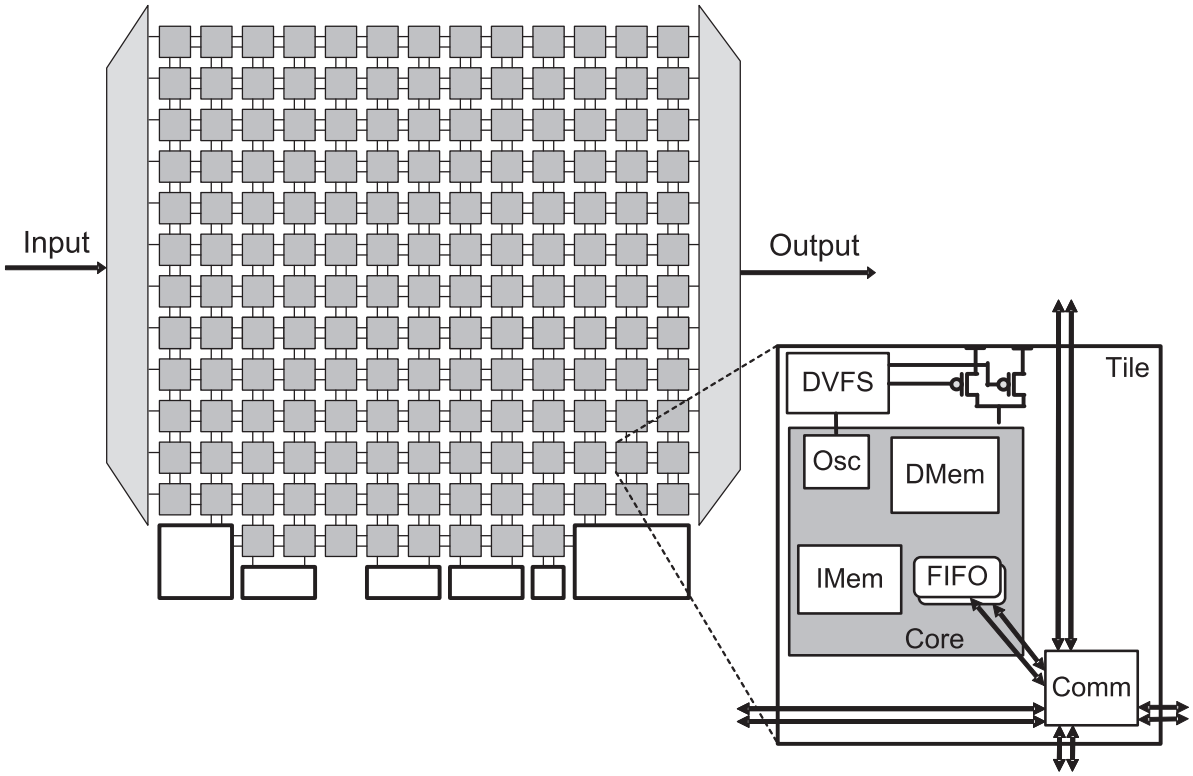


Figure 2.3: Block diagram of the 167-processor computational platform.

A common technique of supplying multiple voltage domains for many-core systems is to integrate on-chip DC-DC converter for each core [92]. However, the overhead of the approach is undesirable if the number of cores increases beyond a few. As a result, it is assumed that there are multiple parallel global power grids in the targeted architecture, and each core can operate at one of several voltage/frequency levels which are controlled by off-chip voltage regulators. As shown in Fig. 2.3, each core can run at one of the two global supply voltages. In this paper, it is assumed that there are multiple discrete global voltage levels (could be more than two) for the whole chip. The per-core DVFS module is capable of controlling the oscillator of each core and selecting the optimal voltage from global voltage supplies based on its running frequency. On the other hand, the global voltage supplies can be adjusted by tuning the off-chip voltage regulators. Without losing generality, in the following of the dissertation, both the number of global voltage supplies and the number of cores can be scaled as necessary.

2.3.3 Programming Methodology on AsAP

Fig. 2.4 illustrates the programming methodology on fine-grained many-core processors. There are basically four steps.

The application is first implemented with a sequential programming language, like C, C++ and Matlab. The program model is based on traditional general-purpose processors with shared memory. The program is used to verify the correctness of the implementation.

Then the sequential program is partitioned into multiple parallel tasks. Each task runs on its own piece of memory, and communicates through message passing interfaces (MPI) with other tasks. Similar as the sequential program model, the input data flows through all the tasks to generate outputs. However, all the tasks can run in parallel that would improve the performance significantly. The partition in this step is still naive that neither performance requirements nor hardware resource limitations (like instruction memory, the number of available cores and so on) is taken into consideration.

After the partition, the tasks are assigned to different logical cores. Based on the performance requirement, hardware resource limitation, and the data flow dependency between tasks, one or more tasks could be assigned to a single core; on the other hand, an individual task could be partitioned further and assigned to multiple cores if necessary [93].

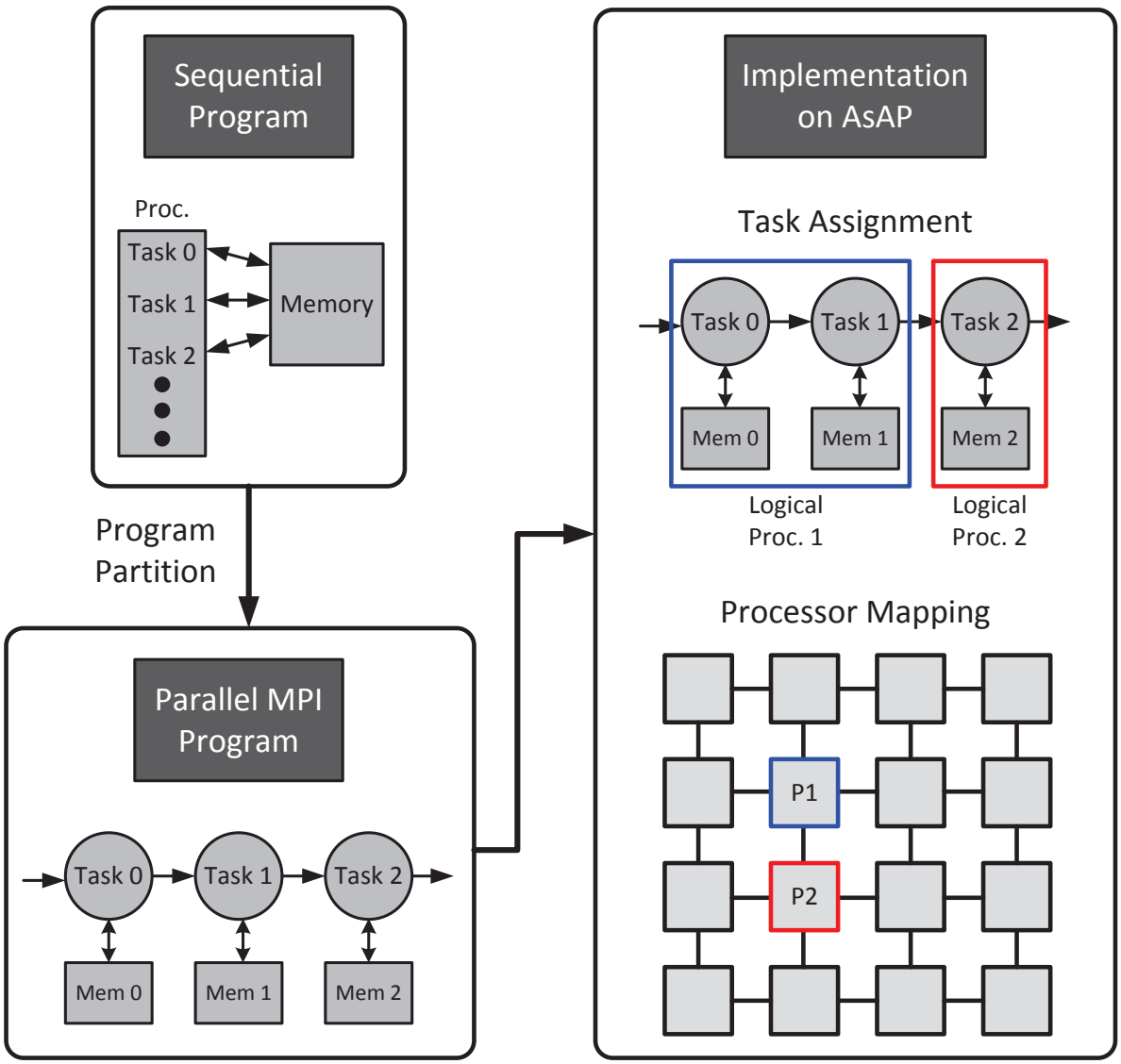


Figure 2.4: Programming methodology on fine-grained many-core processors.

The final step is to map logical cores on the targeted many-core platform. For example, three basic steps are required for the mapping on AsAP [94]:

1. Individual programs on each logical core are written in either C or assembly language.
2. The inputs and outputs of different logical cores are interconnected using a configuration file or a GUI mapping tool [95].
3. The AsAP C compiler, assembler, and automatic mapping tool compile the programs, and produce a configuration file to load the programs into the physical cores on the chip.

Chapter 3

Parallel AES Engines for Many-Core Processor Arrays

3.1 Introduction

With the development of information technology, protecting sensitive information via encryption is becoming more and more important to daily life. In 2001, the National Institute of Standards and Technology (NIST) selected the Rijndael algorithm as the Advanced Encryption Standard (AES) [96], which replaced the Data Encryption Standard (DES) [97]. Since then, AES has been widely used in a variety of applications, such as secure communication systems, high-performance database servers, digital video/audio recorders, RFID tags and smart cards.

To satisfy different applications' requirements, numerous hardware implementations of AES have been reported. Verbauwhede et al. described the first AES implementation on silicon, which can provide a 2.29 Gbps throughput with a non-pipeline architecture [98]. Mukhopadhyay et al. improved their AES system to 8 Gbps with pipelining [99], which is a common technique used to enhance the performance of a system [100]. The first AES implementation with a throughput over 10 Gbps was proposed by applying T-box [101], which is a combination of the *SubBytes*, *ShiftRows* and *MixColumns* phases in the AES algorithm [102]. Furthermore, the area-throughput trade-offs of fully pipelined AES processors with throughputs between 30 to 70 Gbps have been presented [103]. Recently, Mathew et al. implemented a 53 Gbps AES accelerator in 45 nm CMOS technology [104]. Besides application specific integrated circuit (ASIC) designs, configurable hardware is another

choice for AES implementations. For example, there are several FPGA implementations that achieve a throughput approximately 20 to 30 Gbps [105, 106, 107] by applying loop unrolling and pipelining. Recently, Qu et al. demonstrated a 73.7 Gbps AES system on a Xilinx XC5VLX85 chip running at 570 MHz [108].

Although hardware implementations generally offer higher throughput and better energy efficiency than software designs, they are difficult to upgrade and adapt for future possible protocol changes. Moreover, ASIC designs are very time consuming and costly. For example, it takes generally 18 to 24 months for a full custom ASIC product and costs approximately 50 million USD to design [109]. One advantage of the Rijndael algorithm is that it is not only fit for hardware implementations, but also suitable for efficient software designs. Matsui et al. proposed a bitslice AES implementation on Intel Core 2, which achieves a 9.2 clock cycles per byte throughput for a data chunk longer than 2048 bytes, equaling 1.85 Gbps when the core is running at its maximum frequency of 2.13 GHz [110]. The bitslice technique was first proposed by Biham for fast DES implementation on a software platform with a word size longer than 16 bits [111]. Bernstein et al. investigated the opportunities of reducing instruction count and cycles by combining different instructions together for various architectures [112]. Both bitslice and specific sets of instructions from SSSE3 (Supplemental Streaming SIMD Extensions 3 [113]) are utilized to enhance the performance of Intel Core i7 920 as high as 6.92 clock cycles per byte [114]. Besides pure general software AES implementations, the Intel AES-NI utilizes specialized hardware to support six AES instructions, and achieves a throughput of 1.28 clock cycles per byte [115]. There is also a trend to use GPUs (Graphic Processing Units) and DSP processors to implement the AES algorithm. Wollinger et al. compared different encryption algorithms on a TMS320C6X processor and achieved a 14.25 clock cycles per byte [116]. Manavski presented an AES implementation with a peak throughput of 8.28 Gbps on a GeForce 8800 GTX chip when the input data block is longer than 8 MB [117].

This chpater presents various software implementations of the AES algorithm with different data and task parallelism granularity, and shows that AES implementations on a fine-grained many-core system can achieve high performance, throughput per unit of chip area and energy efficiency compared to other software platforms. Both the online and offline key expansion process for each implementation model are discussed. The reminder of this chapter is organized as follows. Section 3.2 introduces the AES algorithm. In Section 3.3, various implementations are analyzed by

synchronous dataflow (SDF) models, mapped and measured on the targeted platform. Section 3.4 presents the area optimization methodology and compares the area efficiency among different implementations. Section 3.5 compares the energy efficiency. Section 3.6 compares our work with other software designs. Finally, Section 3.7 concludes the chapter.

3.2 Advanced Encryption Standard

AES is a symmetric encryption algorithm, and it takes a 128-bit data block as input and performs several rounds of transformations to generate output ciphertext. Each 128-bit data block is processed in a 4-by-4 array of bytes, called the *state*. The *round key* size can be 128, 192 or 256 bits. The number of rounds repeated in the AES, N_r, is defined by the length of the *round key*, which is 10, 12 or 14 for key lengths of 128, 192 or 256 bits, respectively. Fig. 3.1 shows the AES encryption steps with the key expansion process. For encryption, there are four basic transformations applied as follows:

1. *SubBytes:* The *SubBytes* operation is a non-linear byte substitution. Each byte from the input state is replaced by another byte according to the substitution box (called the S-box). The S-box is computed based on a multiplicative inverse in the finite field GF(2^8) and a bitwise affine transformation.

2. *ShiftRows:* In the *ShiftRows* transformation, the first row of the *state* array remains unchanged. The bytes in the second, third and forth rows are cyclically shifted by one, two and three bytes to the left, respectively.

3. *MixColumns:* During the *MixColumns* process, each column of the *state* array is considered as a polynomial over GF(2^8). After multiplying modulo $x^4 + 1$ with a fixed polynomial $a(x)$, given by

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \tag{3.1}$$

   the result is the corresponding column of the output state.

4. *AddRoundKey:* A *round key* is added to the *state* array using a bitwise exclusive-or (XOR) operation. *Round key*s are calculated in the key expansion process. If *Round key*s are calculated on the fly for each data block, it is called AES with online key expansion. On the other

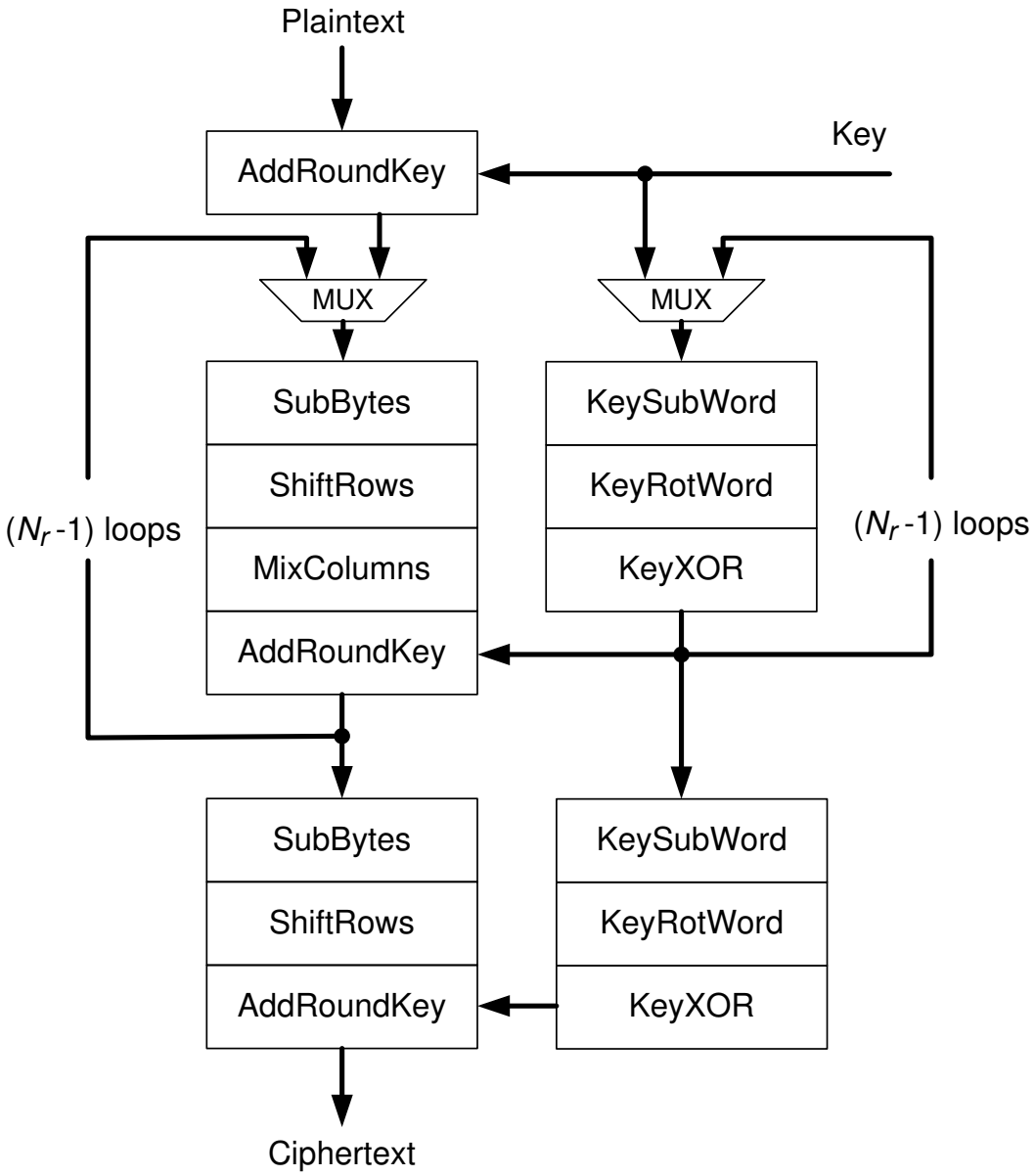


Figure 3.1: Block diagram of AES encryption.

hand, for most applications, the encryption keys do not change as frequently as data. As a result, *round key*s can be calculated before the encryption process, and kept constant for a period of time in local memory or registers. This is called AES with offline key expansion. In this paper, both the online and offline key expansion AES algorithms are examined.

Similarly, there are three steps in each key expansion round.

1. *KeySubWord:* The *KeySubWord* operation takes a four-byte input word and produce an output word by substituting each byte in the input to another byte according to the S-box.

2. *KeyRotWord:* The function *KeyRotWord* takes a word $[a_3, a_2, a_1, a_0]$, performs a cyclic permutation, and returns the word $[a_2, a_1, a_0, a_3]$ as output.

3. *KeyXOR:* Every word $w[i]$ is equal to the XOR of the previous word, $w[i-1]$, and the word Nk positions earlier, $w[i-Nk]$. Nk equals 4, 6 or 8 for the key lengths of 128, 192 or 256 bits, respectively.

The decryption algorithm applies the inverse transformations in the same manner as the encipherment. As a result, only the encryption algorithm is considered in this work for simplicity, since the decipherment yields very similar results.

3.3 AES Implementations on AsAP

In this section, we present 16 different complete and fully-cunctional AES ciphers. The throughput of each design is measured from simulations on a cycle-accurate Verilog RTL model of the actual silicon chip.

Table 3.1 shows the execution delays of different processors on a single AsAP2 chip. For example, *MixColumns-16* executes the *MixColumns* process on a whole 16-byte data block, while *MixColumns-4* performs on a single 4-byte column. The execution time of *MixColumns-4* is more than one fourth of the delay of *MixColumns-16* due to programming overhead on AsAP. Similarly, *SubBytes-16* requires 132 clock cycles to process a 16-byte data block, and it takes 10 clock cycles for *SubBytes-1* to substitute one byte. In our proposed implementations, the key expansion process is divided into two processing units, *KeySubWord* and *KeySchedule*. Each *KeySchedule* processor contains two steps of the key expansion process, *KeyRotWord* and *KeyXOR*.

Table 3.1: Execution delays of processors on AsAP2. Each data block is a 4-by-4 byte array.

Processors	Execution Delays on AsAP2	IMEM Usage
SubBytes-1	10 cycles/byte	9%
SubBytes-4	40 cycles/four bytes	9%
SubBytes-16	132 cycles/block	10%
ShiftRows	38 cycles/block	18%
MixColumns-16	266 cycles/block	63%
MixColumns-4	70 cycles/column	31%
AddRoundKey	22 cycles/block	18%
KeySubWord	56 cycles/block	13%
KeySchedule	60 cycles/block	22%

In the following subsections, we present the eight AES implementations with online key expansion in detail, since the offline implementations can be derived by removing the cores used for key expansion from the online designs. For simplicity, we focus on the situation with a 128-bit key and $N_r = 10$ in this paper, and the impact of different key lengths to our designs is discussed in detail in Subsection 3.3.9.

3.3.1 One-task One-processor

The most straightforward implementation of an AES cipher is to apply each step in the algorithm as a task in the dataflow diagram as shown in Fig. 3.2(a). Then, each task in the dataflow diagram can be mapped on one processor on the targeted many-core platform. This implementation is called one-task one-processor (OTOP). For simplicity, all of the execution delay (shown in Table 3.1), input rates, and output rates in the following dataflow diagrams are omitted. Since the key expansion is processing in parallel with the main algorithm, the throughput of the OTOP implementation is determined by the nine ($N_r - 1 = 9$) loops in the algorithm. The OTOP implementation requires 10 cores on AsAP as shown in Fig. 3.2(b). The throughput of the OTOP implementation is 3582 clock cycles per data block, equaling 223.875 clock cycles per byte.

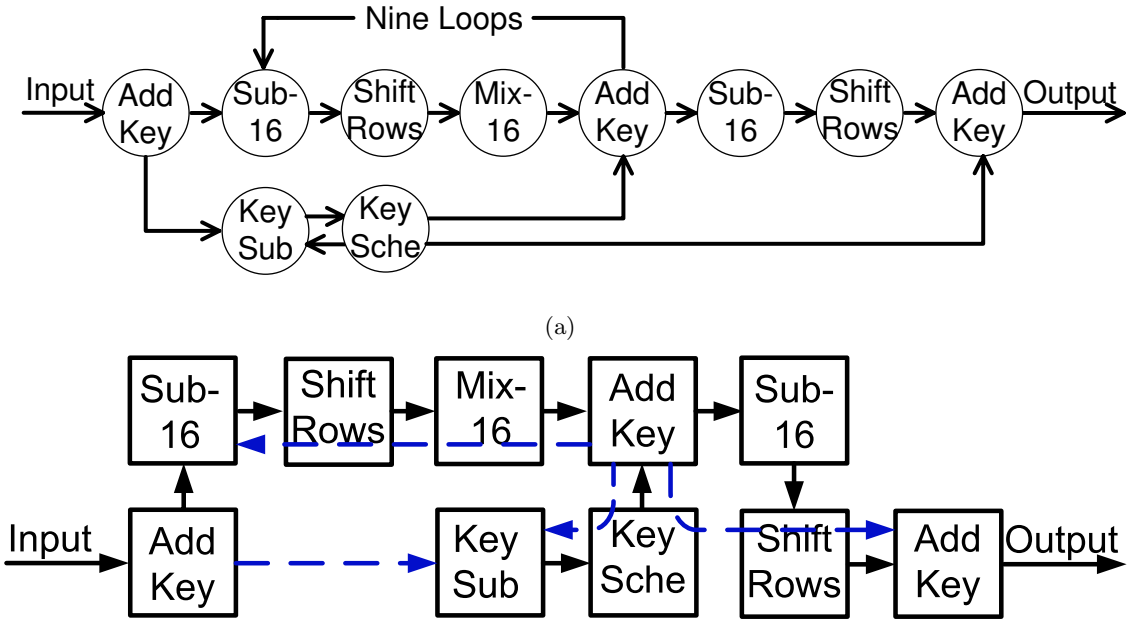


Figure 3.2: One-task One-processor (OTOP) (a) dataflow diagram and (b) 10 cores AsAP mapping.

3.3.2 Loop-unrolled Nine Times

To enhance the AES cipher's throughput, we apply loop unrolling to the OTOP model and obtain the Loop-unrolled Nine Times dataflow diagram as shown in Fig. 3.3(a). The loop unrolling breaks the dependency among different loops and allows the nine loops in the AES algorithm to operate on multiple data blocks simultaneously. To improve the throughput as much as possible, we unroll the loops in both the AES algorithm and the key expansion process by $N_r - 1$ and N_r times, which equals nine and ten, respectively. After loop unrolling, the throughput of the AES implementation is increased to 266 cycles per data block, equaling 16.625 cycles per byte. The mapping of the Loop-unrolled Nine Times model is shown in Fig. 3.3(b), which requires 60 cores.

3.3.3 Loop-unrolled Three Times

To achieve a moderate throughput with fewer cores, we could unroll the main loops in the AES algorithm by S times (S is divisible by $N_r - 1$), instead of $N_r - 1$ times. For this example, the nine loops in the AES algorithm could be split into three blocks, and each block loops three times. The dataflow diagram and mapping of the Loop-unrolled Three Times implementation are

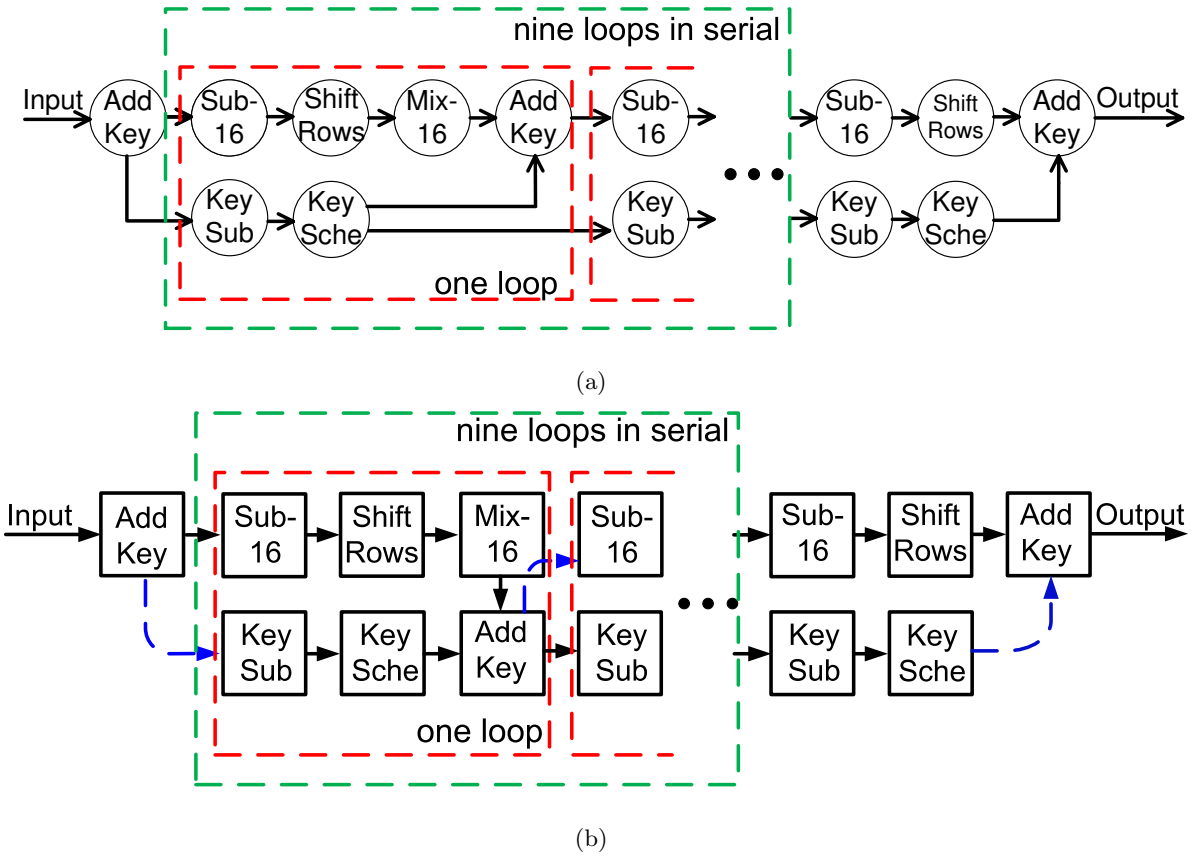
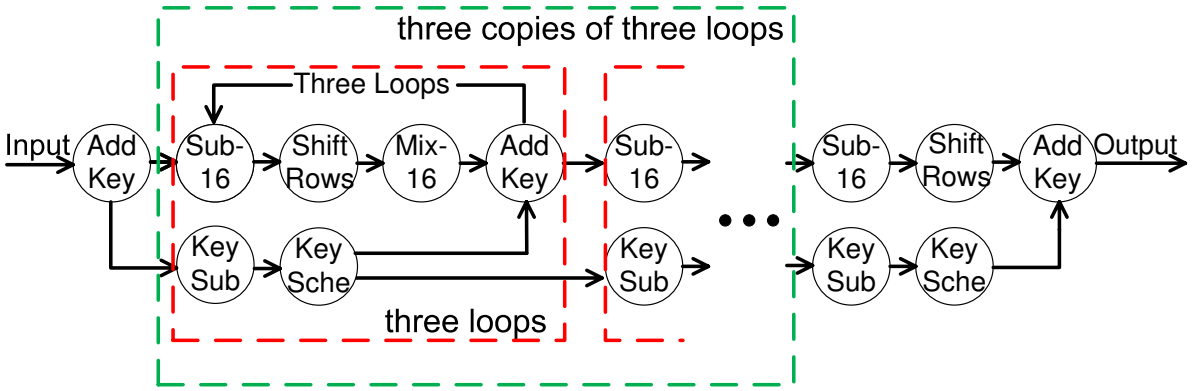
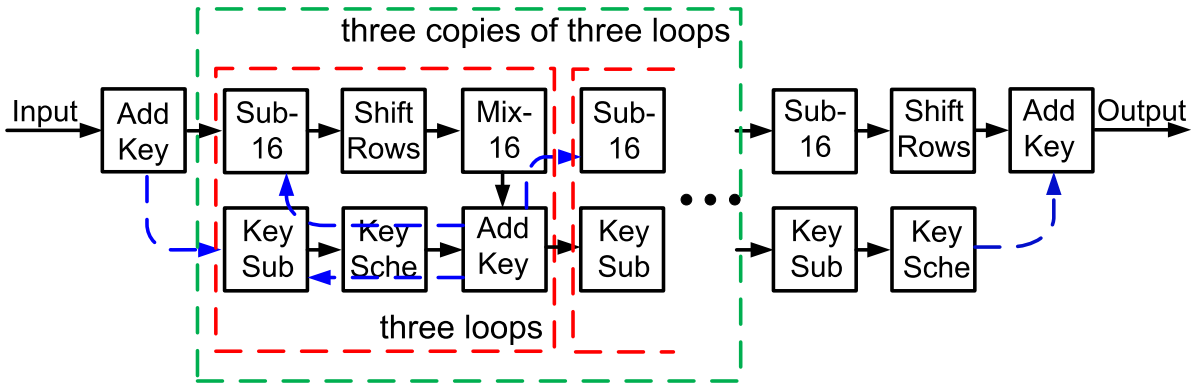


Figure 3.3: Loop-unrolled Nine Times (a) dataflow diagram and (b) 60 cores AsAP mapping.



(a)



(b)

Figure 3.4: Loop-unrolled Three Times (a) dataflow diagram and (b) 24 cores AsAP mapping.

shown in Fig. 3.4(a) and Fig. 3.4(b), respectively. Compared to the OTOP model, the throughput is improved to 1098 cycles per data block, which equals 68.625 cycles per byte; while the mapping requires 24 cores, 36 fewer than the Loop-unrolled Nine Times implementation.

3.3.4 Parallel-MixColumns

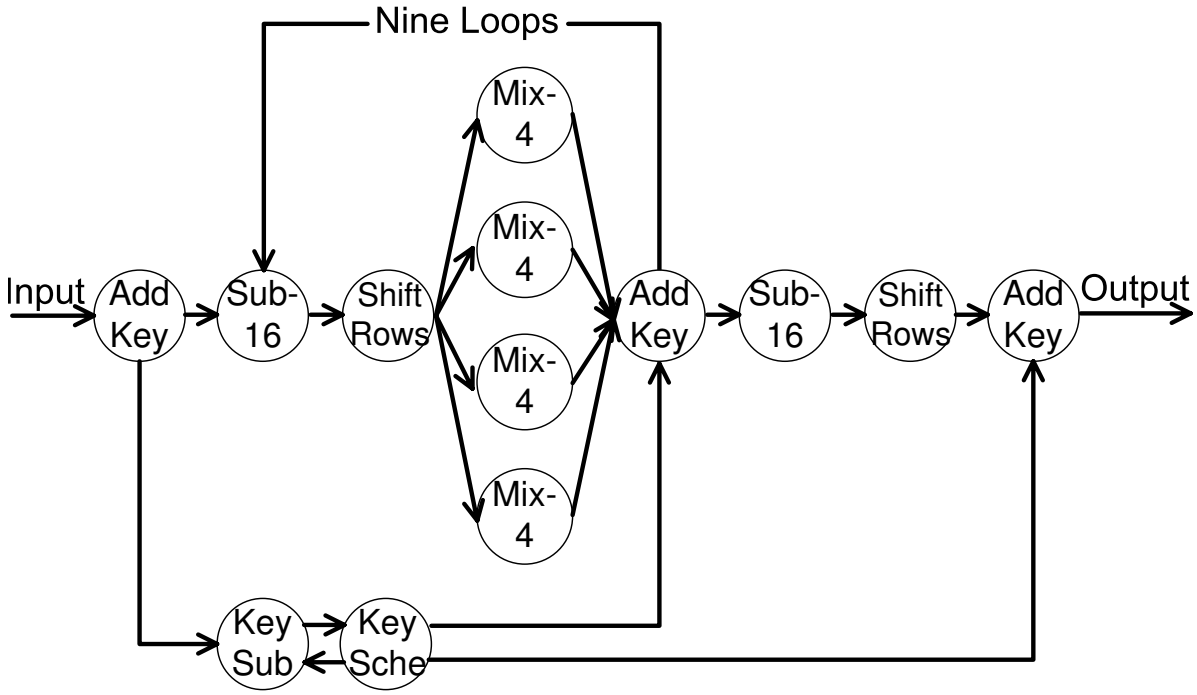
Besides loop unrolling, another way to increase the throughput of the OTOP model is to reduce the main loop's latency in the AES algorithm. In a single loop, the execution delay of *MixColumns-16* results in 60% of the total latency. Each *MixColumns-16* operates on a four-column data block, and the operation on each column is independent. Therefore, each *MixColumns-16* processor can be replaced by four *MixColumns-4*s. Each *MixColumns-4* actor computes only one column rather than a whole data block. As a result, the throughput of the Parallel-MixColumns implementation is increased to 2180 cycles per block, equaling 136.25 cycles per byte. The dataflow diagram and mapping of the Parallel-MixColumns model are shown in Fig. 3.5(a) and Fig. 3.5(b).

Each core on our targeted computational platform can only support two statically configured input ports. Three cores, each called *MergeCore*, are used to merge the four data streams from *MixColumns-4*s into one stream for *AddRoundKey*.

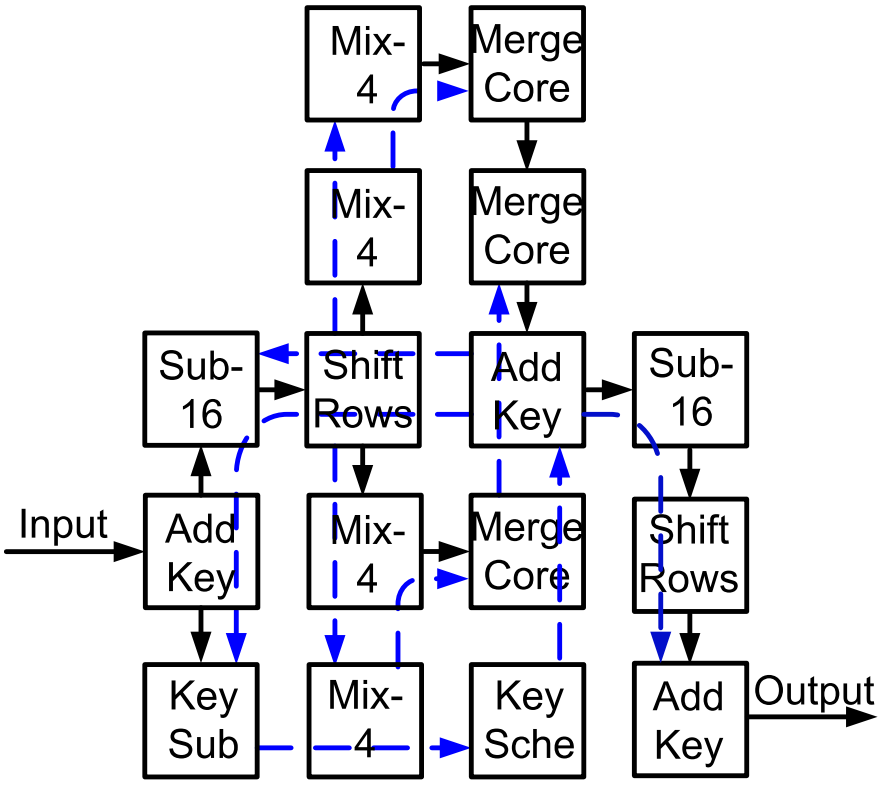
The dependence among bytes in one column diminishes the performance improvement for further parallelization. For instance, if we parallelize one *MixColumns-4* into two *MixColumns-2*s, the effective execution delay of the *MixColumns* process is reduced to 64 cycles from 70 cycles. This saves only 6 cycles while it requires eight more processors (four extra *MixColumns* cores and four extra *MergeCore*s). Therefore, further parallelization on the *MixColumns* process would impair the area and energy efficiency of the entire system without significant performance improvement.

3.3.5 Parallel-SubBytes-MixColumns

In the Parallel-MixColumns implementation, *SubBytes-16* requires 132 cycles to encrypt one data block, which contributes the largest execution delay in one loop. In order to increase the throughput further, we parallelize one *SubBytes-16* into four *SubBytes-4*s, which is shown in Fig. 3.6(a). In this implementation, each *SubBytes-4* processes 4 bytes rather than 16 bytes in one data block. The effective execution delay of the *SubBytes* process is decreased to 40 cycles per block, only around one fourth as before. Therefore, the throughput of the Parallel-SubBytes-MixColumns

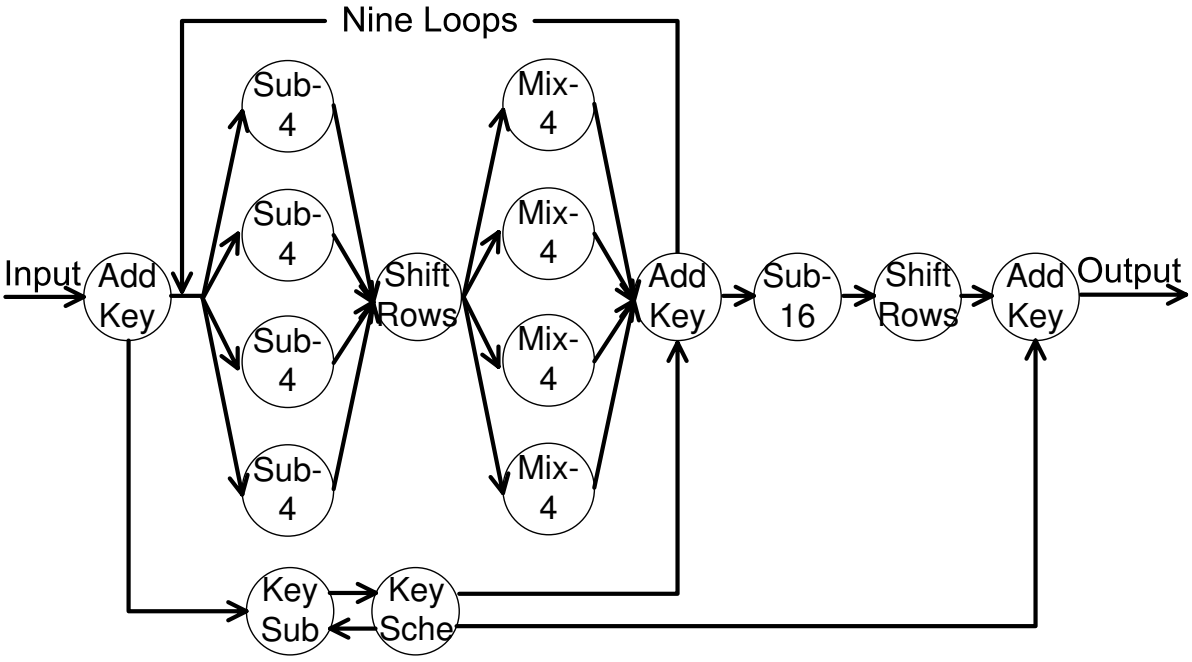


(a)

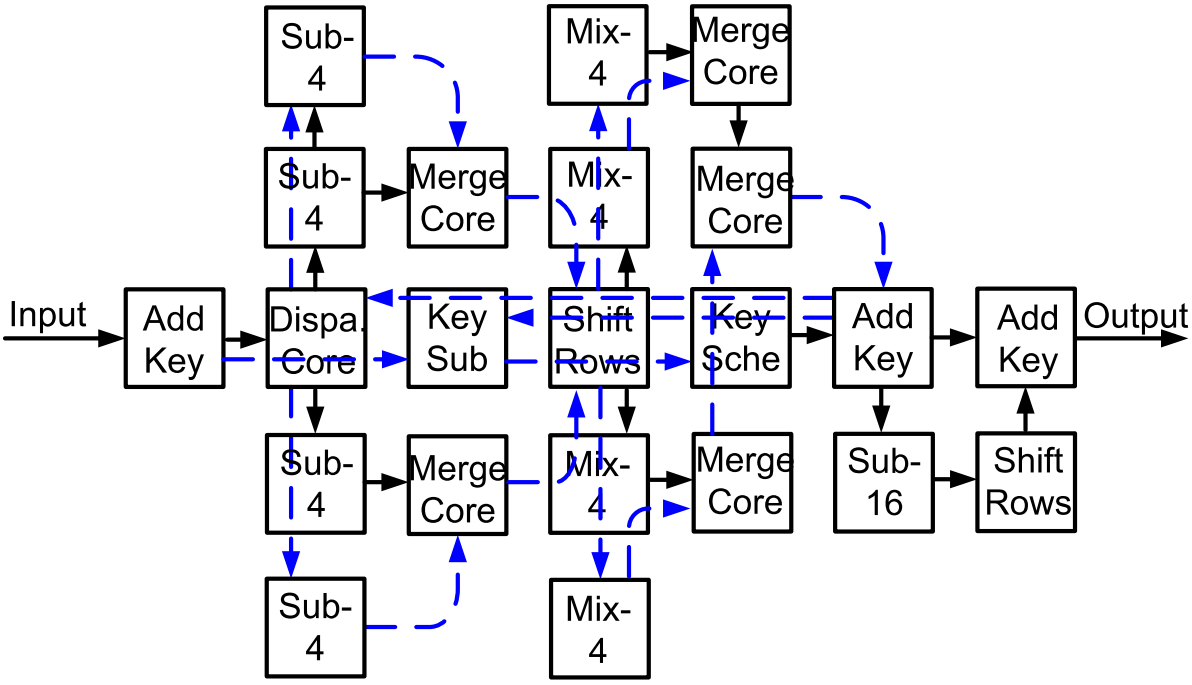


(b)

Figure 3.5: Parallel-MixColumns (a) dataflow diagram and (b) 16 cores AsAP mapping.



(a)



(b)

Figure 3.6: Parallel-SubBytes-MixColumns (a) dataflow diagram and (b) 22 cores AsAP mapping.

model is increased to 1350 cycles per block, equaling 84.375 cycles per byte. The mapping graph of the Parallel-SubBytes-MixColumns implementation on AsAP shown in Fig. 3.6(b) requires 22 cores.

Instead of parallelizing *SubBytes-16* into four *SubByte-4*s, we can replace it with 16 *SubBytes-1*s. The effective execution delay of the *SubBytes* process is reduced to 10 cycles. As a result, the latency of one-loop decreases to 120 cycles. Therefore, the throughput of the cipher is increased to 67.5 cycles per byte. However, it requires seven additional cores dedicated to communication (four *MergeCore*s and three *DispatchCore*s), which impair the area and energy efficiency of the implementation.

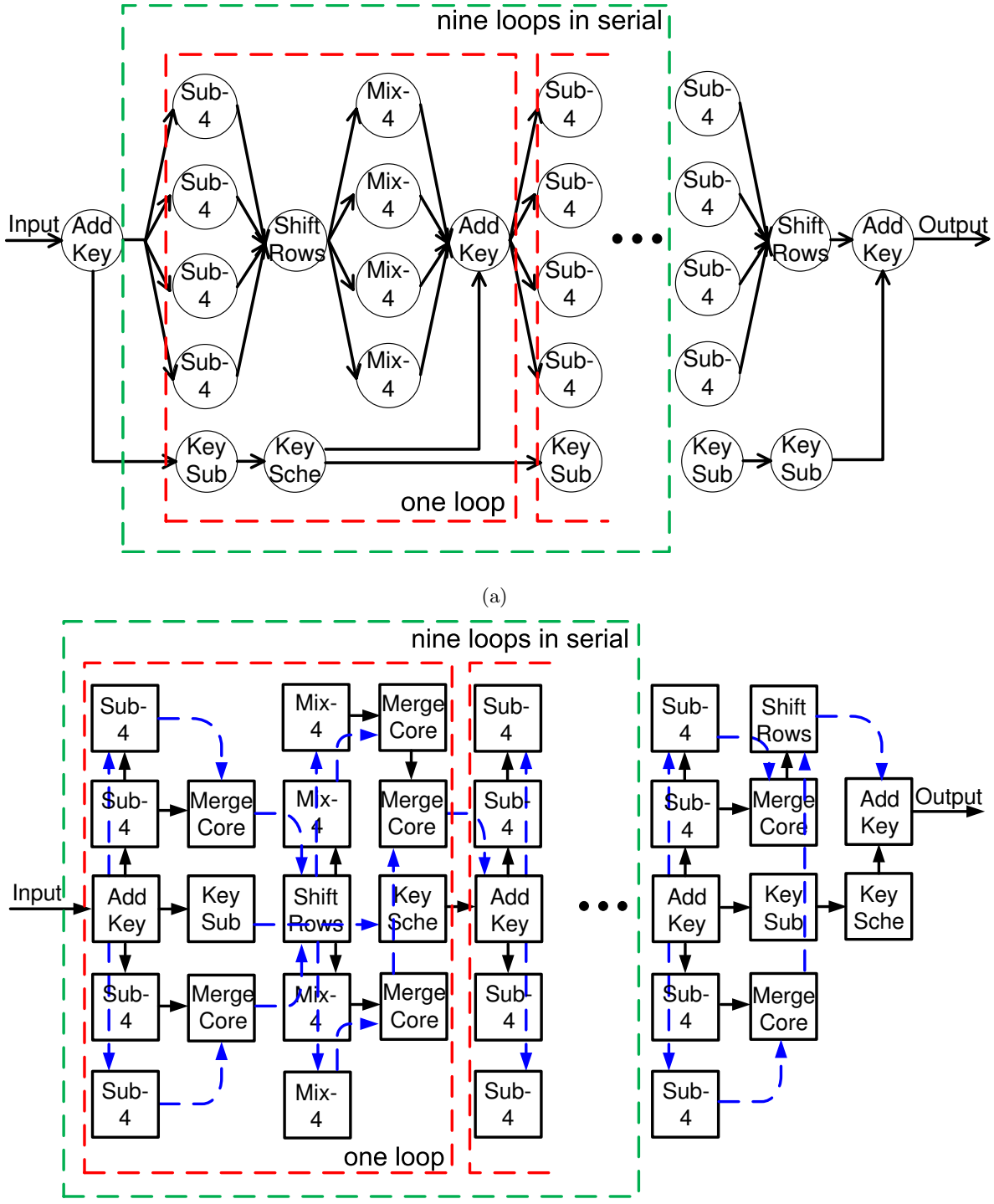
3.3.6 Full-parallelism

The Full-parallelism AES implementation combines the Parallel SubBytes MixColumns model and loop unrolling. The dataflow diagram and the mapping of the Full-parallelism model are shown in Fig. 3.7(a) and Fig. 3.7(b). As expected, the throughput of this design is the highest among all of the models introduced in this paper since it employs most data and task parallelism. The throughput of the Full-parallelism model is 70 cycles per block, equaling 4.375 cycles per byte. It also requires 164 cores, which is the largest implementation of all.

In the Full-parallelism model, the *MixColumns-4* processors are the throughput bottlenecks which determine the performance of the cipher. Therefore, parallelizing the *SubBytes* process with more than four processors would only increase the area and power overhead without any performance improvement.

3.3.7 Small

The Small model implements an AES cipher on AsAP with the fewest processors. As shown in Fig. 3.8, it requires at least eight cores to implement an AES cipher with online key expansion process, since each core on AsAP has only a 128×32-bit instruction memory and a 128×16-bit data memory. The throughput of the Small model is 2678 cycles per data block, which equals 167.375 cycles per byte.



(a)

(b)

Figure 3.7: Full-parallelism (a) dataflow diagram and (b) 164 cores AsAP mapping.

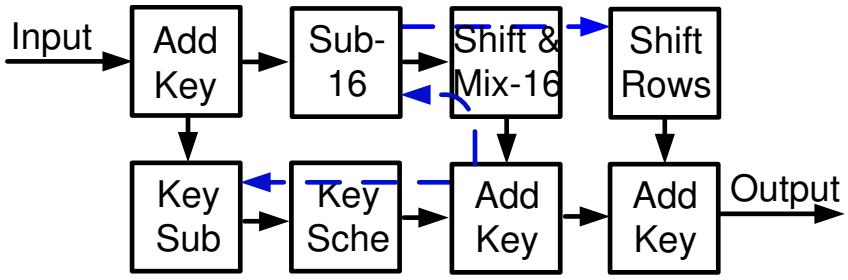


Figure 3.8: 8 cores AsAP mapping of the Small implementation.

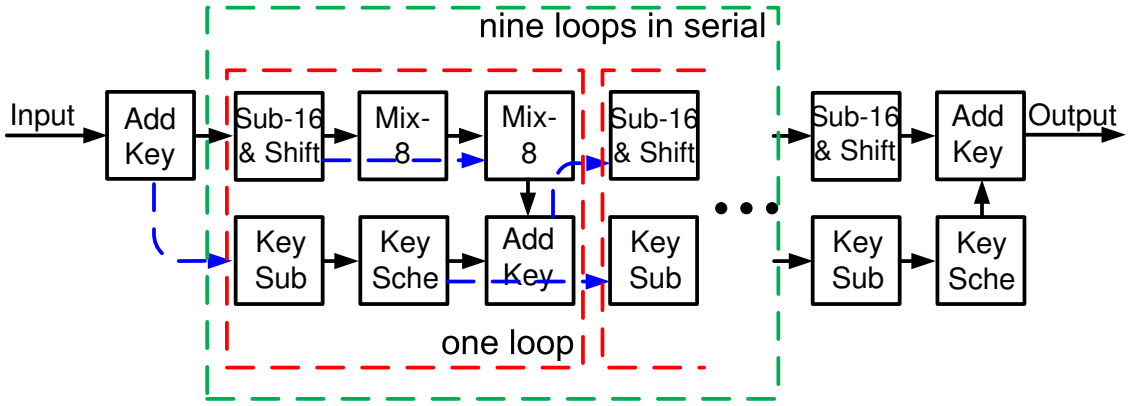


Figure 3.9: 59 cores AsAP mapping of the No-merge-parallelism implementation.

3.3.8 No-merge-parallelism

In contrast to the Small model, the No-merge-parallelism model exploits as much parallelism as possible without introducing any cores dedicated to communication, including *MergeCore*s and *DispatchCore*s. The mapping graph of the No-merge-parallelism implementation on AsAP is shown in Fig. 3.9. To speed up the implementation, loop unrolling is applied in this model. Each *MixColumns-16* is divided into two *MixColumns-8*s, which helps reduce the effective delay of the *MixColumns* process. In order to eliminate additional communication processors and simplify the routing, we combine the *SubBytes* and the *ShiftRows* stages in one core. This implementation requires 59 cores, and has a throughput of 152 cycles per block, equaling 9.5 cycles per byte.

3.3.9 Designs with Longer Keys

As introduced in Section 3.2, besides the 128-bit key, the AES algorithm also supports key lengths of 192 bits and 256 bits. Encrypting with longer keys results in two major areas of additional computation. Firstly, the number of loops in the AES algorithm is increased. Secondly, the key expansion cores require more clock cycles to process round keys.

For the designs without loop-unrolling (Small, OTOP, Parallel-MixColumns and Parallel-SubBytes-MixColumns), no extra cores are required. These mappings operate with longer keys by increasing the number of round loops, N_r, and reprogramming the key expansion related cores. The throughputs of these designs are decreased due to the increased number of N_r rounds.

For the designs with loop-unrolling, additional cores are added depending on the number of rounds required. For example, 12 and 24 more cores are required for the No-merge-parallelism designs with a 192-bit and 256-bit key, respectively. The throughputs of the Loop-unrolled and the No-merge-parallelism are kept the same as before, which is determined by the *MixColumns* operation. On the other hand, for the Full-parallelism implementation, the throughput is decreased since the bottlenecks of the system are shifted from the *MixColumn-4* processors to the key expansion cores, due to the overhead of processing longer keys.

Due to the significant effort required, 192-bit and 256-bit designs are not implemented in this work.

3.4 Area Efficiency Analysis

Area is a significant metric in system design. Less area means less silicon, therefore less cost. From a many-core processor perspective, area is represented by the number of cores required to implement applications. Smaller area translates into fewer used cores and leaves more opportunities for dealing with other applications on the same platform simultaneously. To evaluate the area efficiency between various AES implementations, a metric called *ThroughputPerCore* is defined as the ratio between the throughput of each design to the number of cores used to implement it,

$$ThroughputPerCore = \frac{Throughput}{Number\ of\ Cores} \tag{3.2}$$

3.4.1 Area Optimization Methodology

Before comparing area efficiency among different AES implementations, area optimization is applied to all of the models without impairing performance. In this subsection, the area optimization methodology is illustrated through a detailed example of minimizing the number of cores used by the Full-parallelism model. As shown in Fig. 3.7(b), there are 17 cores in one loop of the Full-parallelism mapping, including five communication-dedicated cores, which are used for routing only. And the final round operation requires 11 cores. Therefore, the number of cores utilized for the un-optimized Full-parallelism model is $(N_r - 1) \times N_{one-loop} + N_{last-round} = 9 \times 17 + 11 = 164$.

Two optimization steps are applied to the Full-parallelism model. First, since the *ShiftRows* process is only byte-rotation, alternating the sequence of the *SubBytes* and the *ShiftRows* stages would not affect encryption results. However, this alternation reduces two *MergeCore*s for each loop. As a result, 18 cores are reduced from the Full-parallelism model. Secondly, the throughput of the Full-parallelism model is 70 cycles per block, which is determined by the operation delay of *MixColumns-4*s. Any actors with less execution delay would not impair the performance of the system. Therefore, a processor fusion of the *ShiftRows* in the Nth loop and the *AddRoundKey* in the $N-1$th loop can reduce one more core for each loop, while keeping the same throughput since these new combination processors take only 60 cycles to process one data block. The dataflow diagram and mapping of the optimized Full-parallelism model are shown in Fig. 3.10(a) and Fig. 3.10(b), respectively.

In summary, without losing any performance, the number of cores required by the online Full-parallelism model is decreased by approximately 16% to 137.

3.4.2 Area Efficiency Comparison

Based on the optimization methods discussed above, the number of cores utilized for each implementation is optimized as follows:

1. Small: Optimization methods are not applicable.
2. OTOP: The *SubBytes* and *ShiftRows* processors in the last round are fused into one processor, saving one processor.

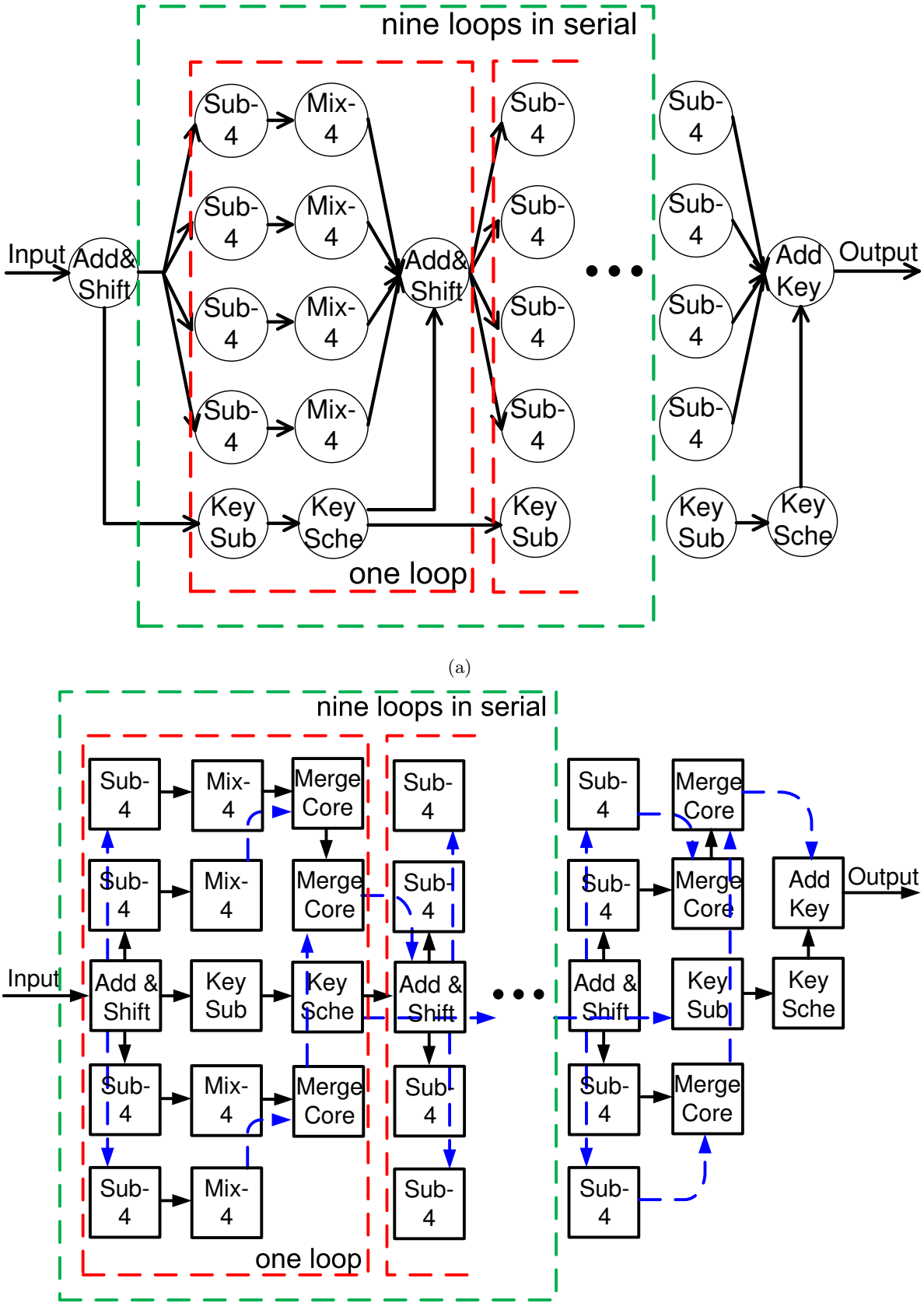


Figure 3.10: Optimized Full-parallelism (a) dataflow diagram and (b) 137 cores AsAP mapping.

3. Parallel-MixColumns: The *SubBytes* and *ShiftRows* processors in the last round are fused into one processor, saving one processor.

4. Parallel-SubBytes-MixColumns: The sequence of the *SubBytes* and the *ShiftRows* stages is alternated, which saves three processors. The *SubBytes* and *ShiftRows* processors in the last round are fused into one processor, saving one more processor.

5. Loop-unrolled Three Times: The *SubBytes* and *ShiftRows* processors in the last round are fused into one processor, saving one processor.

6. Loop-unrolled Nine Times: The *SubBytes* and *ShiftRows* processors in the same round are fused into one processor, which saves 10 processors.

7. No-merge-parallelism: Optimization methods are not applicable.

8. Full-parallelism: The optimization has been discussed in detail in Section 3.4.1

Implementations with Online Key Expansion

The number of cores used for each optimized implementation is shown in Column 3 of Table 3.2. As expected, the Small implementation uses the fewest cores due to its simplicity. On the other hand, the Full-parallelism model occupies 137 cores, exploiting the greatest range of types of data parallelism. As a result, the Full-parallelism implementation requires 17× as many cores as the Small model, while it also gains a 40× throughput increase.

As defined in Equation 3.2, *ThroughputPerCore* is used to compare the area efficiency between different models. The higher the throughput, the better the performance. The fewer the cores used, the smaller the area. As a result, a larger *ThroughputPerCore* ratio shows a higher area efficiency. In Table 3.2, Column 5 shows the *ThroughputPerCore* numbers of various implementations normalized to the Parallel-MixColumns model with online key expansion. The No-merge-parallelism implementation has the highest throughput per core rate, since it avoids any dedicated communication cores and exploits as much parallelism as possible simultaneously. The Full-parallelism and the Loop-unrolled models also offer high throughput per unit of chip area. Although the Small model has a relatively low throughput, it still offers a good area efficiency due to its extremely small area.

Table 3.2: Throughput and the number of cores required by different implementations. Communication cores are used for routing only, including *MergeCores* and *DispatchCores*. All of the throughput per core numbers are normalized to the Parallel-Mixcolumns model with online key expansion.

Implementation	1/Throughput (cycles/byte)	Online Key Expansion			Offline Key Expansion		
		Total Cores	Comm. Cores	Normalized Throughput/Core	Total Cores	Comm. Cores	Normalized Throughput/Core
Small	167.375	8	0	1.53	6	0	2.04
One-task one-processor	223.875	9	0	1.01	7	0	1.30
Parallel-Mixcolumns	136.250	15	3	1	12	2	1.25
Parallel-SubBytes-Mixcolumns	84.375	18	3	1.35	15	2	1.61
Loop-unrolled Three Times	68.625	23	0	1.29	15	0	1.99
Loop-unrolled Nine Times	16.625	50	0	2.46	30	0	4.10
No-merge-parallelism	9.500	59	0	**3.65**	39	0	**5.52**
Full-parallelism	**4.375**	**137**	**30**	3.41	**107**	**20**	4.37

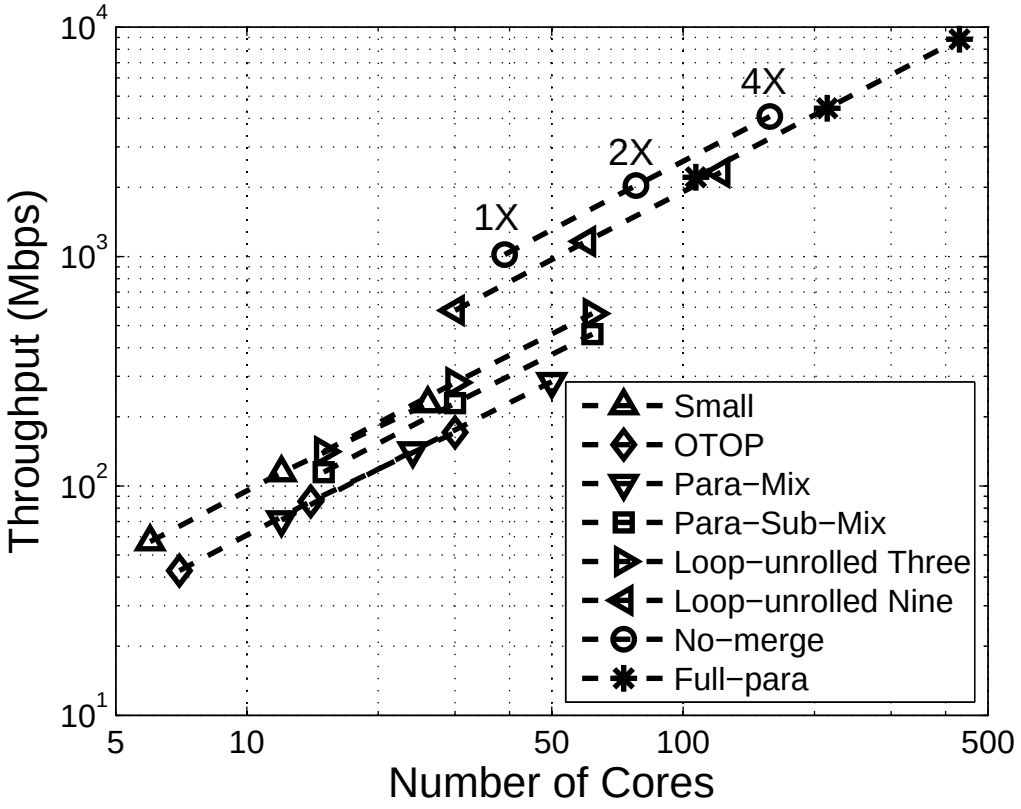


Figure 3.11: Throughput versus the number of cores for the AES implementations with offline key expansion. All processors are running at 1.2 GHz. 1×, 2× and 4× represent the throughput when each implementation is duplicated once, twice and four times, respectively.

Implementations with Offline Key Expansion

Besides the online key expansion AES algorithm, the detailed results of AES with offline key expansion are also shown in Columns 6, 7 and 8 of Table 3.2. The processors used for key expansion process can be eliminated for the AES implementations with offline key expansion, which results in 29% improvement in average throughput per area compared to the implementations with online key expansion.

The throughput versus the number of cores of the eight offline implementations is shown in Fig. 3.11. The throughput is obtained when all processors are running at 1.2 GHz. Besides the basic implementations discussed above, we duplicate each implementation two and four times to scale the throughput and area. On the targeted platform, for any scaled implementation with a 4× duplication, two merge-cores are required to gather the outputs for the subsequent processor by assuming each processor could take only two inputs.

3.5 Energy Efficiency Analysis

In this section, the power consumption and energy efficiency of the previously discussed eight implementations are investigated based on chip measurement results.

3.5.1 Power Numbers from Chip Measurements

Each core on AsAP can operate up to 1.2 GHz at 1.3 V [91]. The maximum frequency and power consumption of cores on AsAP have a near-linear and quadratic dependence on the supply voltage, as shown in Fig. 3.12. The average power dissipation of one core and communication link at 1.3 V and 1.2 GHz is shown in Table 3.3. This supply voltage and clock frequency are used in the power estimation and optimization case study in the next subsection. The table also shows during stalls (i.e. non-operation while the clock is active), the processors still consume a significant portion, approximately 50%, of its active power. The leakage power is decreased to a negligible number when the clock is halted.

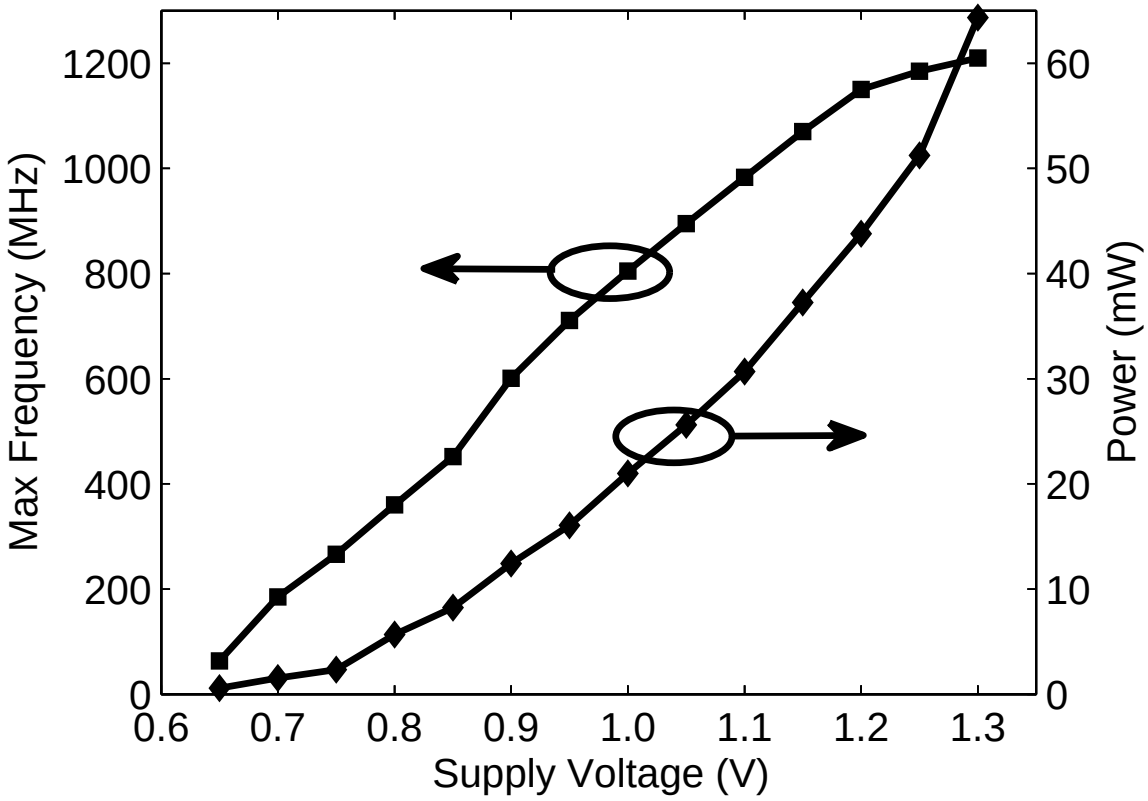


Figure 3.12: Maximum operation frequency and 100% active power dissipation of one core versus supply voltage.

Table 3.3: Average power dissipation measured at 1.3 V and 1.2 GHz [91]

Operation of	100% Active (mW)	Stall (mW)	Leakage (mW)
Processor	59.5	31.0	0.13
Nearest-neighbor comm.	5.9	NA	∼0
Long-distance comm. one tile	12.1	NA	∼0

3.5.2 Power Estimation Methodology

On our targeted platform, each processor has four states: *active*, *NOP with active clock*, *stall with active clock*, and *standby with halted clock*. The *active* mode means that the processor is busy with instruction execution, while the *NOP with active clock* represents the NOP operation in programs due to data and control hazards. Either an empty input or a full output FIFO is capable of halting each processor's clock and causing the processor to sleep in the *standby with halted clock* mode. Finally, the *stall with active clock* is the transition state between *active* and *stall with halted clock*. As a result, the overall power dissipated by all of the processors utilized in any implementation can be derived by:

$$\begin{aligned} P_{Total} = & \sum P_{Exe.i} + \sum P_{Stall.i} \\ & + \sum P_{Leak.i} + \sum P_{Comm.i} \end{aligned} \tag{3.3}$$

where $P_{Exe.i}$, $P_{Stall.i}$, $P_{Leak.i}$ and $P_{Comm.i}$ are the power consumed by computation execution, stalling (including NOP) with active clock, standby with halted clock (leakage only) and communication activities of the i^{th} processor, respectively, and are estimated as follows:

$$\begin{aligned} P_{Exe.i} &= \alpha_i \cdot P_{ExeAvg} \\ P_{Stall.i} &= \beta_i \cdot P_{StallAvg} \\ P_{Leak.i} &= (1 - \alpha_i - \beta_i) \cdot P_{LeakAvg} \end{aligned} \tag{3.4}$$

where P_{ExeAvg}, $P_{StallAvg}$ and $P_{LeakAvg}$ are the average power of processors while performing 100% execution, stalling (including *NOP with active clock* and *stall with active clock*) or halting (leakage only); α_i, β_i and $(1 - \alpha_i - \beta_i)$ are the percentages of execution, stall and standby activities of processor i, respectively. The communication power can be calculated as follows,

$$P_{Comm.i} = \gamma_i \cdot P_{CommNear} + \delta_i \cdot P_{CommLong} \tag{3.5}$$

where γ_i and δ_i are the percentages of communication between neighboring and long-distance processors, respectively. $P_{CommNear}$ is the 100% active power consumed by a link when it is used for communication between neighboring processors, while $P_{CommLong}$ is for long-distance [118].

The optimized Full-parallelism model with offline key expansion is used as an example to illustrate the power estimation methodology discussed above. For the ith processor, its α_i, β_i

Table 3.4: Operation cycles and power consumption of offline key expansion Full-parallelism implementation at 1.3 V and 1.2 GHz.

Processor	Number	Execution Time (cycles)	NOP with AC[a] (cycles)	Stall with AC[a] (cycles)	Stall with HC[b] (cycles)	Nearby Comm. (cycles)	Long-dist. Comm. (cycles)	Execution Power (mW)	Stall Power (mW)	Leakage Power (mW)	Comm. Power (mW)	i^{th} Core Power (mW)
AddKeyShiftRows	10	37	0	18	15	8	8	31.47	7.97	0.03	2.05	40.13
SubByte-1	40	27	12	16	15	4	0	22.97	12.40	0.03	0.34	36.59
MixColumn-4	36	63	7	0	0	2	2	53.59	3.10	0	0.51	57.20
FinalRoundAddKey	1	18	0	40	12	16	0	15.31	17.71	0.02	1.35	21.60
MergeCore	20	10	0	44	16	0	8	8.34	19.49	0.03	1.38	18.63
Total								$\mathbf{3.35 \times 10^3}$	$\mathbf{1.09 \times 10^3}$	**2.12**	**81.4**	$\mathbf{4.52 \times 10^3}$

[a] AC stands for active clock.

[b] HC stands for halted clock.

and $(1-\alpha_i-\beta_i)$ are derived from Columns 3, 4, 5 and 6 of Table 3.4. Furthermore, γ_i and δ_i are obtained from Column 7 and 8. Note that the throughput of the Full-parallelism implementation is 70 cycles per block.

Additionally, if the implementation works under 1.3 V and 1.2 GHz, the power consumed by execution, stalling, standby and communication activities of each processor are listed in Columns 9, 10, 11 and 12. In Column 2, the number of processors with the same operation are listed. Therefore, the total power number can be derived by the following equation and is listed in the last row.

$$P_{Total} = \sum N_i \cdot P_{Total_i} \tag{3.6}$$

where N_i is the number of processors with the ith kind of operation, and P_{Total_i} is the total power dissipated by the ith processor. The total power consumption is 4.52 W with a 2.21 Gbps throughput. The communication power consumed by FIFOs and switches is 81.4 mW, which is 1.8% of the total power, while the leakage power is only 2.12 mW and 0.05% of the total power dissipation.

3.5.3 Energy Efficiency Comparison

The energy efficiency of a system describes how much energy is consumed for processing a specific workload. This metric influences a critical design parameter, battery lifetime, made more important by the increasing popularity of mobile devices. In our discussion, the energy efficiency is defined as the energy dissipated for processing one bit by

$$\begin{aligned} EnergyPerBit_{Vdd} &= Power_{Vdd}/Throughput_{Vdd} \\ &= (Power_{Vdd} \times Delay)/freq_{Vdd} \times 128 \end{aligned} \tag{3.7}$$

where $Power_{Vdd}$ and $freq_{Vdd}$ are the power dissipation and frequency for one model at supply voltage Vdd. $Delay$ represents the number of clock cycles required for processing one data block. Since power has a general relationship with supply voltage and operation frequency as $Power \propto Vdd^2 \cdot f$, from Eq. 3.7, it is expected that $EnergyPerBit_{Vdd} \propto Vdd^2$.

As shown in Fig. 3.13, for the eight offline implementations discussed above, the energy dissipated for processing one bit is nearly quadratically dependent on supply voltage, which is consistent with the theoretical analysis. Furthermore, the no-merge model consumes the least

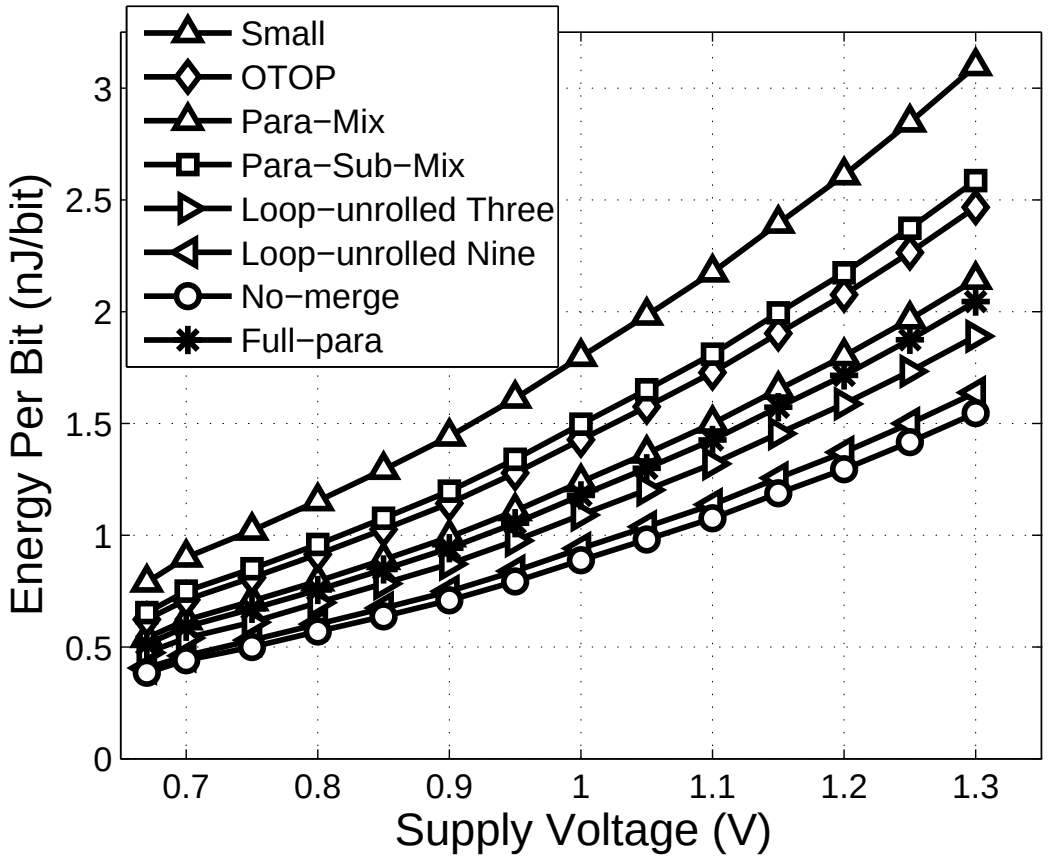


Figure 3.13: Energy consumed for processing one bit of data versus supply voltage. All of the implementations shown in this figure are associated with offline key expansion.

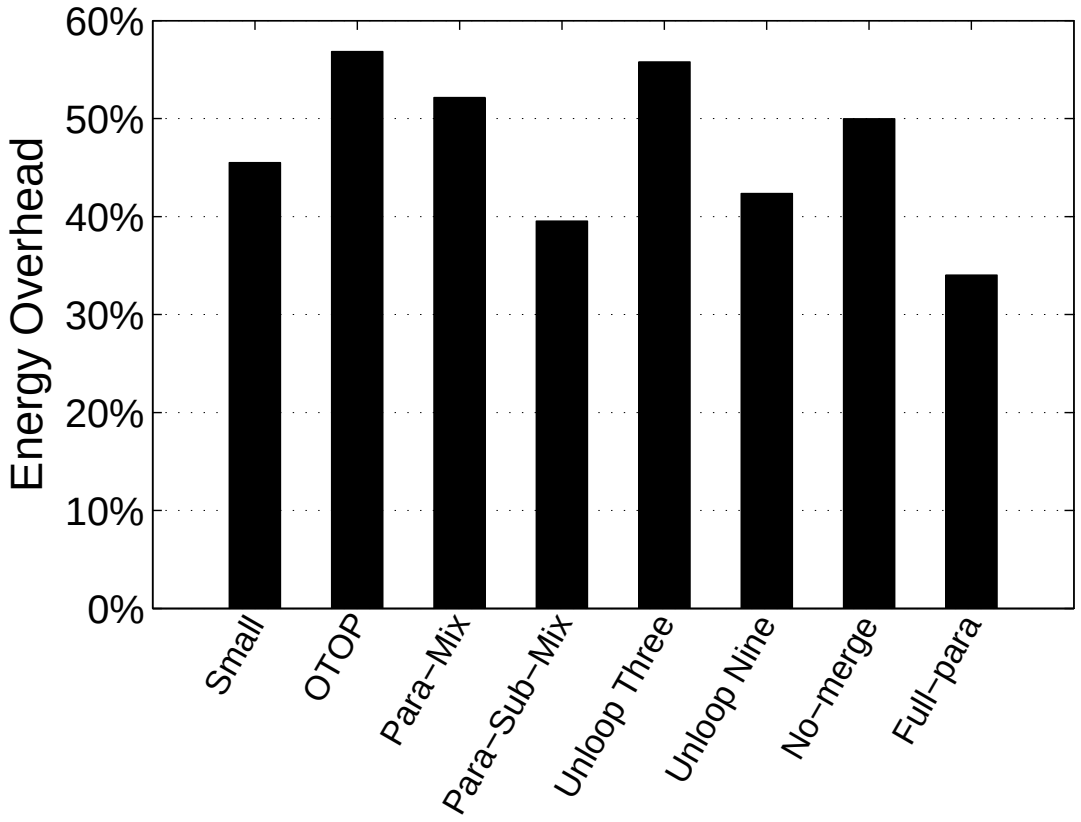


Figure 3.14: Energy overhead of the AES implementations with online key expansion compared with the ones with offline key expansion.

energy to encrypt one bit compared with other implementations, which is from 0.39 to 1.54 nJ/bit depending on the supply voltage and throughput. On the other hand, the Parallel-MixColumns implementation shows the lowest energy efficiency, which consumes approximately 2× the energy to encrypt a data block as the No-merge-parallelism model.

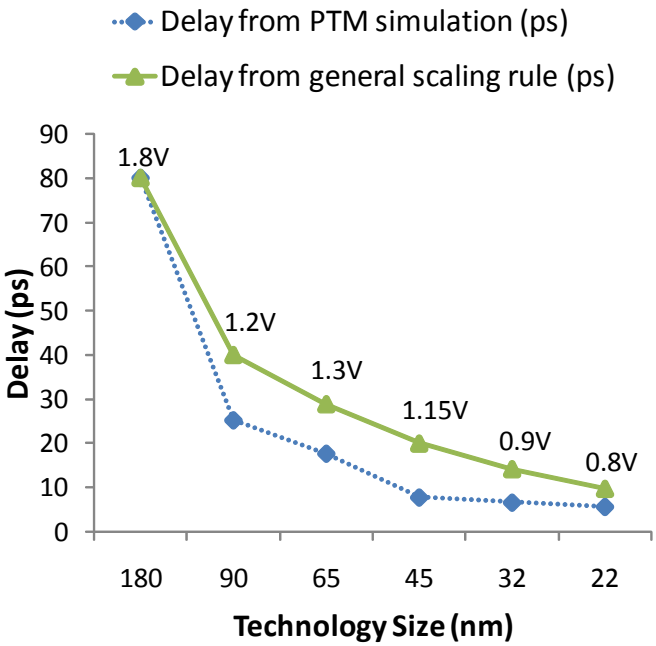
Fig. 3.14 shows that the AES implementations with online key expansion consume 35% to 55% more energy to process same workload, compared to their counterparts with offline key expansion.

3.6 Related Work and Comparison

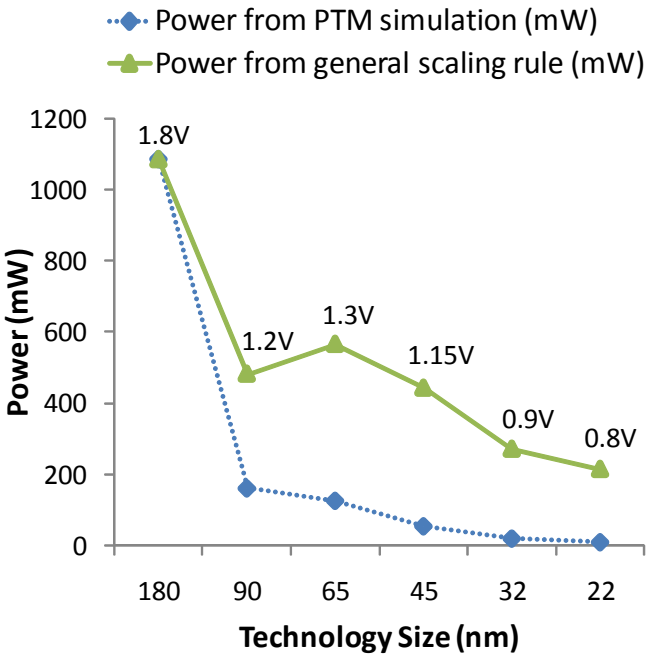
Since the AES ciphers presented in this work are implemented on a programmable platform without any application-specific hardware, we compare our work with other software implementations on programmable processors, and do not compare with implementations that contain or are composed of specialized hardware (e.g., ASICs, ASIPs, FPGAs, etc.). AES hardware implementations have been reported to achieve throughput per area up to tens and even hundreds of Gbps/mm^2 [104] and energy efficiencies in the range of several pJ/bit—they are in an entirely different class both in efficiencies achieved and in the cost and effort required to design.

A comprehensive comparison of the state-of-the-art software AES implementations is summarized in Table 3.5. In order to make a fair comparison, all of the referenced data are scaled to 65 nm CMOS technology with a supply voltage of 1.3 V. The area data are scaled to 65 nm with a $1/(s^2)$ reduction, where s equals the ratio between the minimum feature size of the old technology and 65 nm. The delay and power data are scaled by SPICE simulation results of a fanout-of-4 (FO4) inverter under different technologies and supply voltages with predictive technology model (PTM) [120] as shown in Fig. 3.15.

As discussed in Section 3.4.2, we could always map one of our designs for multiple times to get a higher throughput while possibly introducing a small overhead. Therefore, it is less meaningful to compare the throughput solely of each design. In this section, we use the metrics of throughout per chip area (Mbps/mm^2) and energy per workload bit (nJ/bit) to compare the area efficiency and energy efficiency of various designs. As shown in Table 3.5, compared to the highly optimized AES ciphers on CPUs with bitslice [110], the proposed AES cipher on AsAP has 3.5–9.2 times higher



(a) Delay



(b) Power

Figure 3.15: Delay and power of a FO4 inverter based on SPICE simulation using predictive technology model (PTM) [119]; the general scaling rule assumes a $1/s$ reduction in delay and a $1/(v^2)$ reduction in power where s is the technology scaling factor and v is the voltage scaling factor [43].

Table 3.5: Comparison of AES cipher implementations on different software platforms. The original data are presented with different CMOS technologies and supply voltages. For comparison, area, performance and power consumption are scaled to 65 nm technology with a supply voltage of 1.3 V.

Platform	Method	Tech. (nm)	Area (mm^2)	Vdd (V)	Max Freq. (MHz)	Throughput (cycles/byte)	Power (W)	Scaled Throughput (Mbps)	Scaled Area (mm^2)	Scaled Throughput/Area (Mbps/mm^2)	Scaled Energy/bit (nJ/bit)
Pentium 4 561 [110]		90	112	1.2	3600	16	57.5	2570	58.42	43.99	17.50
Athlon 64 3500 [110]		90	193	1.2	2200	10.6	44.5	2370	101	23.55	14.69
Core2 Duo E6400 [110]	Bitslice	65	111	1.3	2130	9.19	32.5	1854	111	16.70	17.53
Core2 Quad Q6600 (one core)[114]	Bitslice + SSSE3	65	286/2 = 143	1.3	2400	9.32	26.25	2060	143	14.41	12.74
Core2 Quad Q9550 (one core)[114]	Bitslice + SSSE3	45	214/4 = 53.5	1.15	2830	7.59	11.88	1307	112	11.71	21.16
Core i7 920 (one core)[114]	Bitslice + SSSE3	45	263/4 = 65.75	1.15	2668	6.92	16.25	1351	137	9.84	28.00
TI C6201 [116]		180	NA	1.8	200	14.25	NA	509	NA	NA	NA
GeForce 8800 GTX [117]	T-box	90	484	1.2	575	NA	67.5	11800	252	46.82	4.48
This Work AsAP [91]	No-merge offline key expan.	65	6.63	1.3	1210	9.5	**1.58**	1019	**6.63**	**153.70**	**1.55**

throughput per unit of chip area and consumes 9.5–11.3 times less energy to encrypt a fixed amount of data. Besides bitslice, SIMD instructions are applied to improve the throughput and efficiency of AES implementations on CPUs further [114]. Even so, our design on AsAP still has 10.7–15.6 times higher throughput per unit of chip area and 8.2–18.1 times lower energy per bit. The TI DSP C6201 is an 8-way VLIW architecture for high performance DSP applications. The referenced data shows that our design has 2 times higher throughput. The area and power numbers of the TI DSP C6201 are not available, but we believe that AsAP has significantly higher throughput per unit of chip area and energy efficiency due to a much smaller core size.

The AES implementation on GeForce 8800 GTX achieves the highest throughput in the referenced designs, due to its large chip area and the utilization of the T-Box method, which works effectively for SIMD architectures with large memory [117]. However, our design still shows a 3.3 times higher throughput per unit of chip area and 2.9 times higher energy efficiency.

Fig. 3.16 shows that the software AES implementation on AsAP outperforms other software platforms in terms of energy efficiency and performance per area.

3.7 Conclusion

We have presented 16 different AES cipher implementations with both online and offline key expansion on a fine-grained many-core system. Each implementation exploits different levels of data and task parallelism. The smallest design requires only 6 processors, equaling 1.02 mm^2 in a 65 nm fine-grained many-core system. The fastest design achieves a throughput of 4.375 cycles per byte, which is 2.21 Gbps when the processors are running at a frequency of 1.2 GHz. We also optimize the area of each implementation by examining the workload of each processor, which reduces the number of cores used as much as 18%. The design on the fine-grained many-core system achieves energy efficiencies approximately 2.9–18.1 times higher than other software platforms, and performance per area on the order of 3.3–15.6 times higher. Overall, the fine-grained many-core system has been demonstrated to be a very promising platform for software AES implementations.

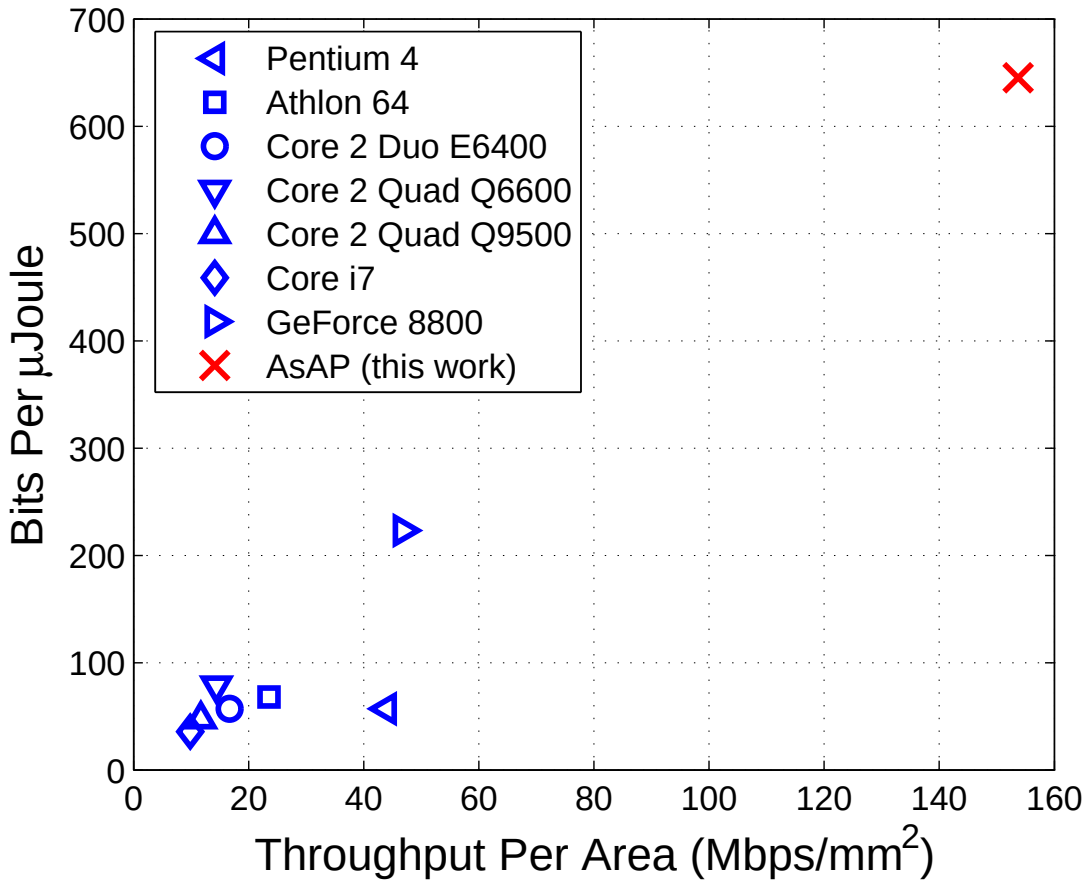


Figure 3.16: Comparison of peak performance per area and workload per unit energy of programmable processors. All numbers are scaled based on the PTM simulation results shown in Fig. 3.15.

Table 3.6: Summary of state-of-the-art ASIC and FPGA AES implementations

ASIC Implementations	Tech. (nm)	Vdd (V)	Freq. (MHz)	Throughput (Mbps)	Power (mW)	Area (mm^2)	Throughput/Area (Gbps/mm^2)	Energy/bit (pJ/bit)
Mathew *et al.* [104]	45	1.1	2100	53,000	125	0.15	353	2.36
Mathew *et al.* [121]	22	0.9	1100	432	13	0.0022	196	31
Zhang *et al.* [122]	40	0.9	1300	494	4.39	0.0043	115	8.85
FPGA Implementations	**Device**		**Freq. (MHz)**	**Throughput (Mbps)**		**Area (slices)**	**Throughput/Area (Mbps/slice)**	
Hodjat *et al.* [105]	XC2VP20-7		168	21500		5177	4.2	
Chang *et al.* [106]	XC2VP2		306	876		156	5.6	
Granado-Criado *et al.* [107]	XC2V6000-6			24920		3576	7.0	
Qu *et al.* [108]	XC5VLX85		576	73737		22994	3.2	

Chapter 4

Scalable Joint Local and Global Dynamic Voltage and Frequency Scaling for Many-Core Systems

4.1 Introduction

With the continuous scaling of CMOS technology, a large number of processing elements (PE) are able to be integrated on a single silicon die, resulting in multi-processor systems-on-chip (MPSoCs). Many-core processors with network-on-chip (NoC) interconnects are promising architectures for high performance energy-efficient computing [91], [24]. As the technology scales, a single chip with 1000+ processors was recently reported [29], [123].

One of the critical challenges for many-core system design is to reduce the power dissipation and improve the energy efficiency of the chip. It is predicted that without introducing novel low power architectures or techniques, more than 50% of the chip has to be turned off due to power concerns when CMOS technology shrinks to 8 nm [21]. Researchers are eager to seek innovative solutions to relieve the "dark silicon" problem and effectively convert transistors to performance. Additionally, processors with high energy efficiency not only save millions of dollars in energy costs for supercomputers and data centers, but also extend the battery life for mobile devices.

As illustrated in Fig. 4.1, due to the diversity of the applications mapped on different

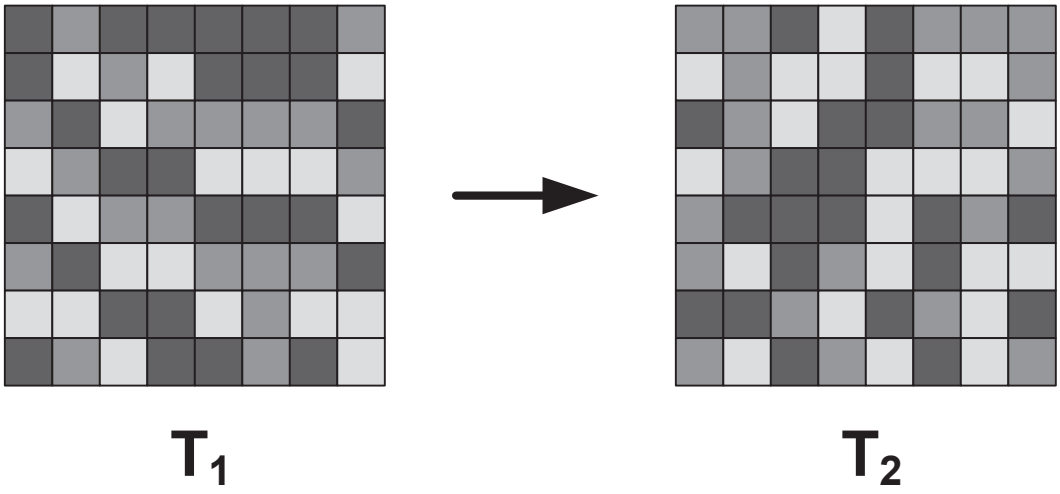


Figure 4.1: Workload varies among cores, and also changes over time in many-core systems.

cores on many-core systems, the workload and performance requirement of each individual core may be different and also changing over time. There are computationally intensive jobs that require processors to run at the highest frequency. On the other hand, there are also non-performance critical jobs which can be computed at a lower frequency while still meeting the performance requirement. Dynamic voltage and frequency scaling (DVFS) exploits the fact that dynamic power is proportional to $V^2 \times freq$, to perform dynamic voltage and frequency scaling in order to provide "just-enough" processor speed to finish the workload under time/performance constraints, while reducing power dissipation. Many-core processors with per-core DVFS are capable of reducing power dissipation significantly by adapting each core's supply voltage and working frequency according to its workload [124], [125].

4.1.1 Related Work

One of the biggest challenges of DVFS implementation is to estimate the workload accurately so that the potential energy saving can be fully extracted without introducing any performance penalty, e.g. missing deadlines or throughput requirements. Chandrakasan et al. proposed a mechanism to scale the voltage and frequency dynamically by analyzing input data dependency and predicting computational workload in the future [67]. Sinha and Chandrakasan proposed a workload prediction algorithm based on adaptive filtering, and compared it with other filtering

schemes in terms of performance and energy saving [68].

For real-time applications, Mosse et al. proposed and analyzed several techniques to dynamically adjust processor speed with slack reclamation [126]. Aydin et al. proposed an intra-task scheduling algorithm for uniprocessor systems, based on both static worst-case execution time (WCET) analysis and online speed speculation [69]. Zhu et al. studied a run-time slack reclamation scheme for tasks sharing a global deadline on multiprocessor systems [70]. Then they extended their work to address the case of aperiodic tasks with dependency constraints [127].

Additionally, several works focused on developing practical DVFS policies that rely on architectural runtime statistics to determine the workload. Marculescu proposed a voltage scaling scheme driven by cache misses [75]. Ghiasi et al. used IPC (instructions per cycle) rate to predict the workload and scale the voltage [76]. Choi et al. presented a regression-based fine-grained DVFS model by monitoring on-chip computation and off-chip memory access cycles [128]. Dhiman and Rosing explored the opportunity by applying an online learning algorithm along with runtime statistics for further power reduction [77].

Recently, various DVFS schemes based on control system theory have been discussed in the literature. All existing work can be broadly divided into two categories, one is based on FIFO (or queue) occupancy, the other is based on FIFO stall information. Wu et al. formally described a nonlinear model of the queue occupancy, and presented a proportional-integral-derivative (PID) controller, which requires detailed analysis of the queue behavior before actual hardware implementation [71]. Alimonda et al. analyzed more complex queue configurations, and developed a non-linear controller which offers better transient and steady-state performance, as compared with linear controllers [72]. Orgas et al. presented an adaptive feedback controller based on state-space models to determine the optimal voltage frequency island (VFI) for different PEs [129]. Garg et al. extended Orgas's work by adopting both local and global state feedback to balance the energy saving and the implementation complexity of the DVFS controller [130]. On the other hand, Choudhary and Marculescu proposed a method which adopts the stall information, rather than the occupancy, of FIFOs. The DVFS controller counts the stall cycles from both the producer and the consumer of a communication link to determine the optimal VFI for PEs in the next control interval [73].

4.1.2 Chapter Organization and Contributions

In this chapter, we propose an online scalable hardware-based joint local and global DVFS solution driven by the workload variations for many-core processors. The local algorithm is used to select the voltage and frequency for each individual core based on its workload, while the global algorithm adjusts the voltage supplies provided for the whole chip according to the workload of all active cores. We also demonstrate that the coordinated local and global technique can improve the energy efficiency significantly under performance constraints.

The major contributions provided by this work are: 1) quantitative analysis and comparison between frequency scaling and voltage dithering; 2) two different local DVFS algorithms based on both occupancy and stall information of FIFOs for both upstream and downstream constrained systems, respectively; and the comparison between the two algorithms in terms of power saving, voltage switching frequency and response delay to workload variation; 3) a global algorithm to choose the voltage settings for many-core processors with a limited number of voltage supplies; and 4) analysis of the sensitivity of power saving improvement brought by the global optimization to different parameters, including the number of global voltage supplies, throughput requirements and the tuning resolution of voltage regulators.

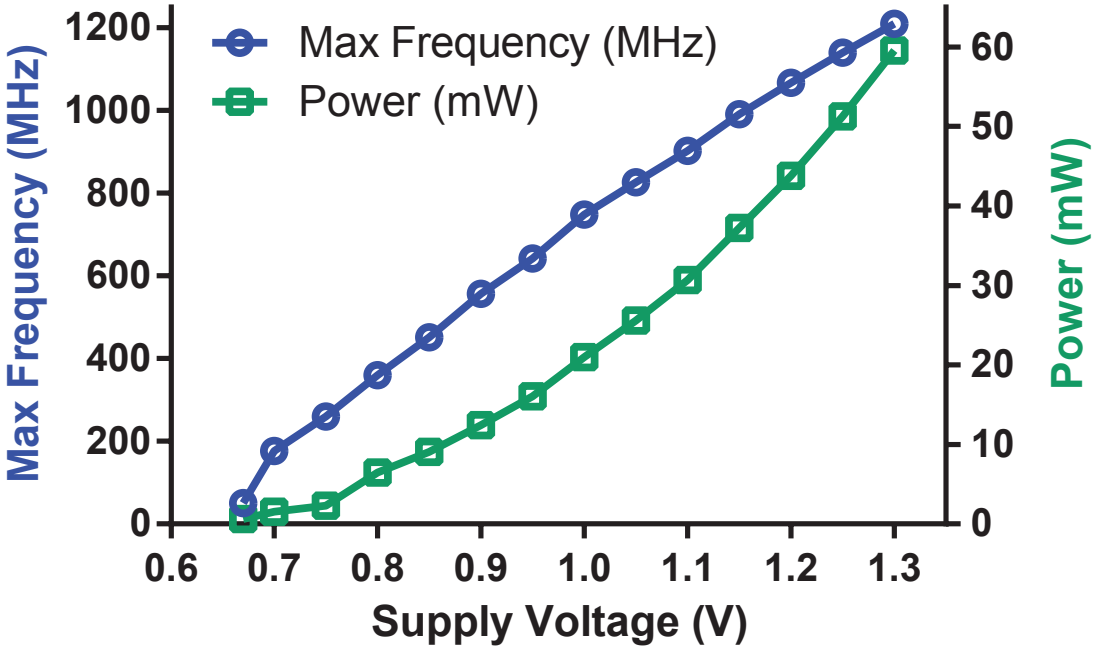
The remainder of this chapter is organized as follows. Section 4.2 provides the background of the targeted many-core system and the power model used in our experiments. Section 4.3 compares frequency scaling and voltage dithering in detail. The proposed local and global algorithms are presented in Section 4.4 and Section 4.5, respectively. Section 4.6 discusses the experiment results with detailed analysis. Finally, Section 4.7 concludes the chapter.

4.2 Preliminaries

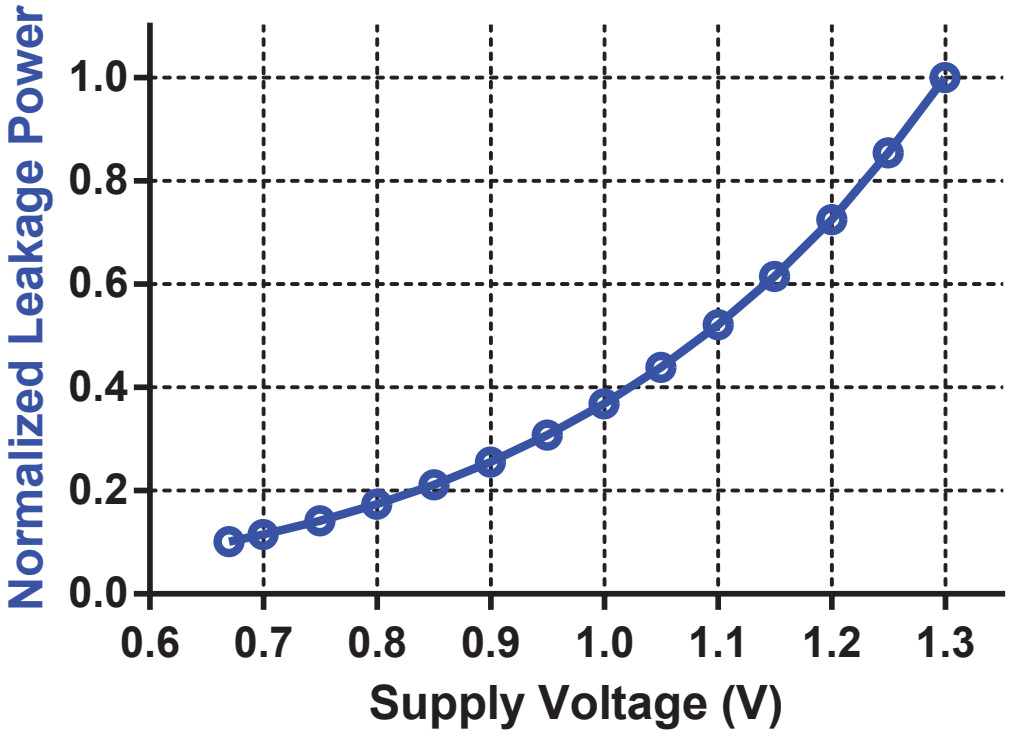
The AsAP2 many-core platform is used in our experiments. The detailed architecture of AsAP2 has been presented in Section 2.3.2. Fig. 2.3 shows the block diagram of AsAP2.

4.2.1 Power Model

The maximum operation frequency and dynamic power dissipation of cores on AsAP have a near-linear and quadratic dependence on the supply voltage, as shown in Fig. 4.2(a). The data



(a) Maximum Frequency and Dynamic Power



(b) Normalized Leakage Power

Figure 4.2: (a) Maximum operation frequency and dynamic power of one core; (b) Normalized leakage power scaling factor.

are measured from the silicon chip, and used for the simulations in the following sections.

Since AsAP was fabricated with a high threshold voltage CMOS process, the leakage power from the chip measurement is negligible. To simulate processors with different leakage ratio $LR = P_{LEAK}/P_{DYN}$ at 1.3 V, the leakage power scaling factors for different voltage supplies are generated from a dummy circuit with ST 65 nm CMOS technology. The dummy circuit is composed of a large number of NAND, NOR and INV gates. Gates have a different number of inputs (1 to 4) and the input states are randomly selected [131]. The normalized leakage power scaling factors are measured from the HSPICE simulation and shown in Fig. 4.2(b).

4.3 Frequency Scaling Versus Voltage Dithering

For systems with N global discrete voltage supplies (V_1, V_2, ..., V_N) in ascending order, there are two major voltage and frequency scaling schemes. The first one is called *limited DVFS*, which prioritizes frequency scaling over voltage scaling. The processor changes its operation frequency based on its workload, and selects the lowest possible supply voltage from the global power grids,

$$V_{core} = \begin{cases} V_i, & \text{if } f(V_{i-1}) < f_{req} \leq f(V_i) \\ V_1, & \text{if } f_{req} \leq f(V_1) \end{cases} \tag{4.1}$$

where $f(V_i)$ is the maximum frequency allowed under V_i, and f_{req} is the desired working frequency.

In contrast of prioritizing frequency scaling, *voltage dithering* only allows processors to run at the maximum available frequency under a specific voltage, which represents by a voltage frequency (VF) pair (V_i, f_i). Processors swing periodically between the VF pairs above and below their desired operation frequency to achieve the performance requirements [132]. For example, if the available normalized frequencies are 0.75 and 0.5, while the desired rate is 0.6, the core would spend 40% of processing time at the frequency of 0.75, and 60% at 0.5. In general, voltage dithering can be described as:

$$V_{core} = \begin{cases} P \times V_i + (1-P) \times V_{i-1}, & \text{if } f_{i-1} < f_{req} \leq f_i \\ V_1, & \text{if } f_{req} \leq f_1 \end{cases} \tag{4.2}$$

where P and $(1-P)$ are the percentage of processing time at which the local V_{core} is assigned to

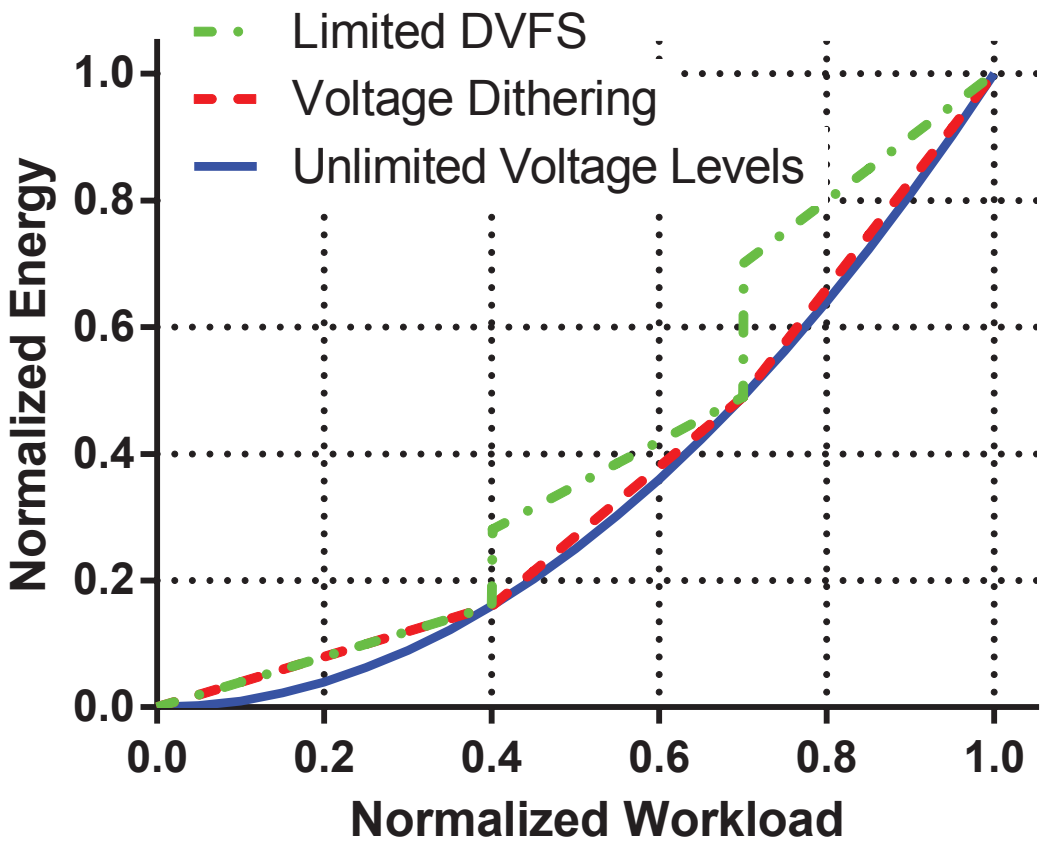


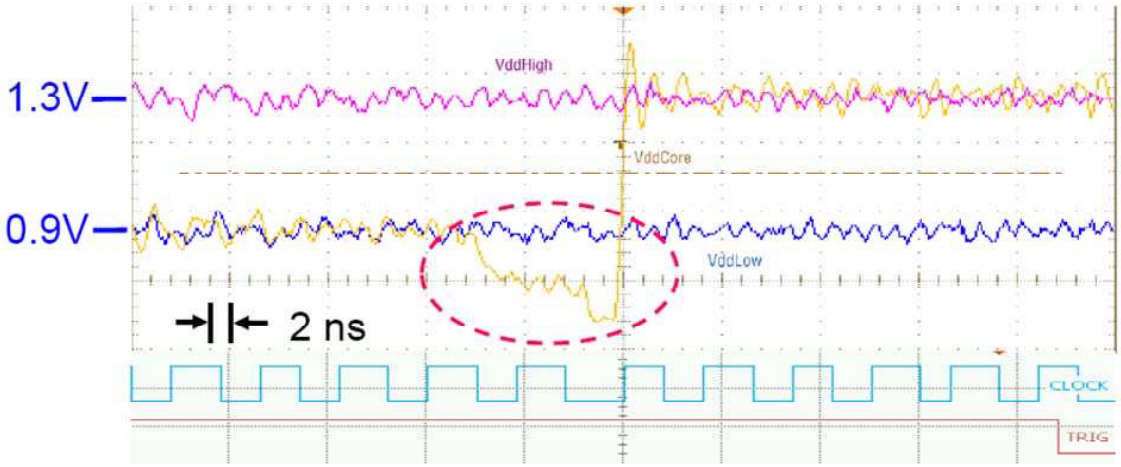
Figure 4.3: Normalized energy versus workload of frequency scaling with three voltages, voltage dithering with three VF pairs and theoretically ideal case with arbitrary levels of voltage domain.

V_i and V_{i-1}, respectively. P is defined according to the desired core's frequency as

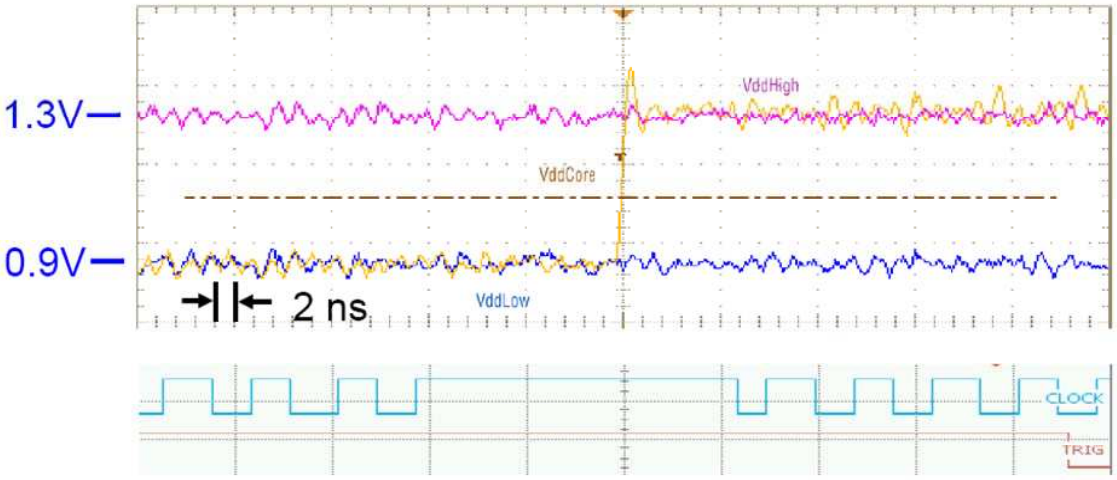
$$P = \frac{f_{req} - f_{i-1}}{f_i - f_{i-1}} \quad \text{if } i > 1 \tag{4.3}$$

As shown in Fig. 4.3, voltage dithering is more energy efficient than limited DVFS, and very close to the optimal scenario with infinite voltage domains theoretically. However, voltage dithering introduces two major overheads compared to the limited DVFS in practical implementations. First, switching between voltage supplies takes more clock cycles compared to adjusting the clock frequency on the same power grid. As shown in Figure 4.4, the clock is required to be halted to eliminate the voltage droop and noise during voltage switches. As a result, voltage switches introduce extra computation delay as overhead. Additionally, the processor requires extra energy charge whenever V_{core} switches from a lower voltage rail to a higher one.

To investigate the overhead brought by voltage switches, we define the energy consumption



(a)



(b)

Figure 4.4: Voltage noise measurements on AsAP during voltage switches with (a) active clock and (b) halted clock.

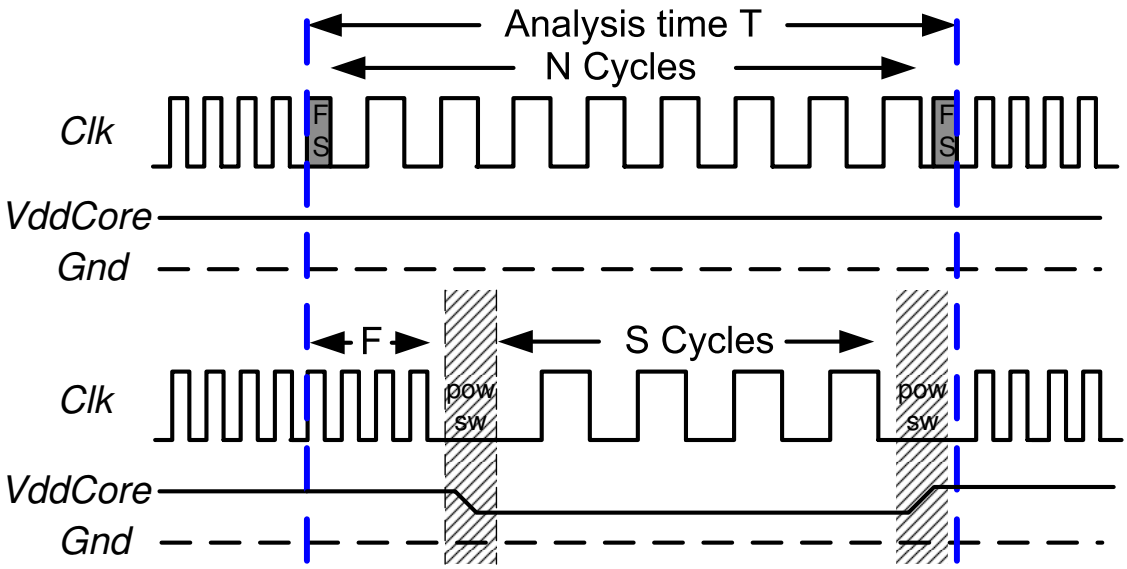


Figure 4.5: Comparison between limited DVFS with frequency scaling only (upper) and voltage dithering (lower).

E_{freq} for frequency scaling (limited DVFS) and E_{dith} for voltage dithering, respectively. As shown in Fig. 4.5, the frequency scaling method lowers down the clock frequency for N cycles, then bumps it back to the original clock frequency. The V_{core} stays on V_{high} during the whole analysis period T. Therefore, the total energy consumption during T is obtained as

$$E_{freq} = \alpha C_L V_{high}^2 N + I_{leak}(V_{high}) V_{high} T \tag{4.4}$$

where α is the switching (or activity) factor, C_L is the load capacitance for each core, and I_{leak} is the leakage current depends on the voltage supply.

On the other hand, the voltage dithering method would stay on V_{high} for F cycles, switch to V_{low} for S cycles and then switch back to V_{high}. The total energy consumption is formulated as

$$\begin{aligned} E_{dith} =& \alpha C_L V_{high}^2 F + I_{leak}(V_{high}) V_{high} \frac{F}{f_{high}} \\ & \alpha C_L V_{low}^2 S + I_{leak}(V_{low}) V_{low} \frac{S}{f_{low}} \\ & + C_P (V_{high} - V_{low}) V_{high} + 2E_{sw} \end{aligned} \tag{4.5}$$

where C_P is the power line capacitance for power grid switches, and E_{sw} is the energy consumption during each voltage switch. Considering the clock is halted during voltage switches, only leakage

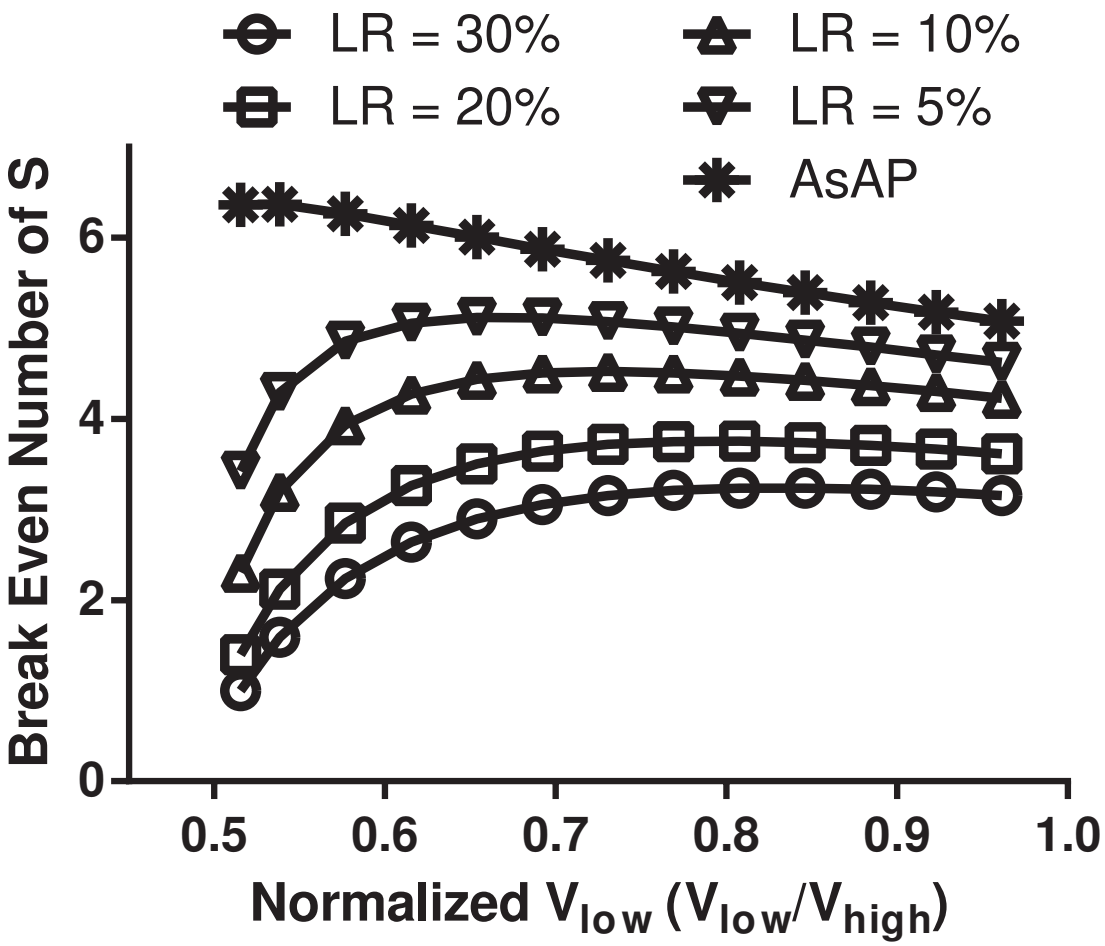


Figure 4.6: Break even number of S for E_{dith} less than E_{freq} with different leakage power ratio. $V_{high} = 1.3$ V, $\alpha = 0.1$.

power contributes to E_{sw}. Assume the core is connected with V_{high} half of the switching time, and with V_{low} for the other half, E_{sw} can be calculated as follows

$$E_{sw} = \frac{1}{2}(I_{leak}(V_{high})V_{high}T_{sw} + I_{leak}(V_{low})V_{low}T_{sw}) \tag{4.6}$$

where T_{sw} is the time duration with stalled clock for each voltage switch. Additionally, both the number of operation cycles and the total operation time should be identical for the above two cases.

$$N = F + S \tag{4.7}$$

$$T = \frac{F}{f_{high}} + \frac{S}{f_{low}} + 2T_{sw} \tag{4.8}$$

As discussed in Subsection 2.3.2, each AsAP core is capable of switching between voltage supplies by simply toggling power gates. Therefore, C_P is approximately equal to C_L. Additionally,

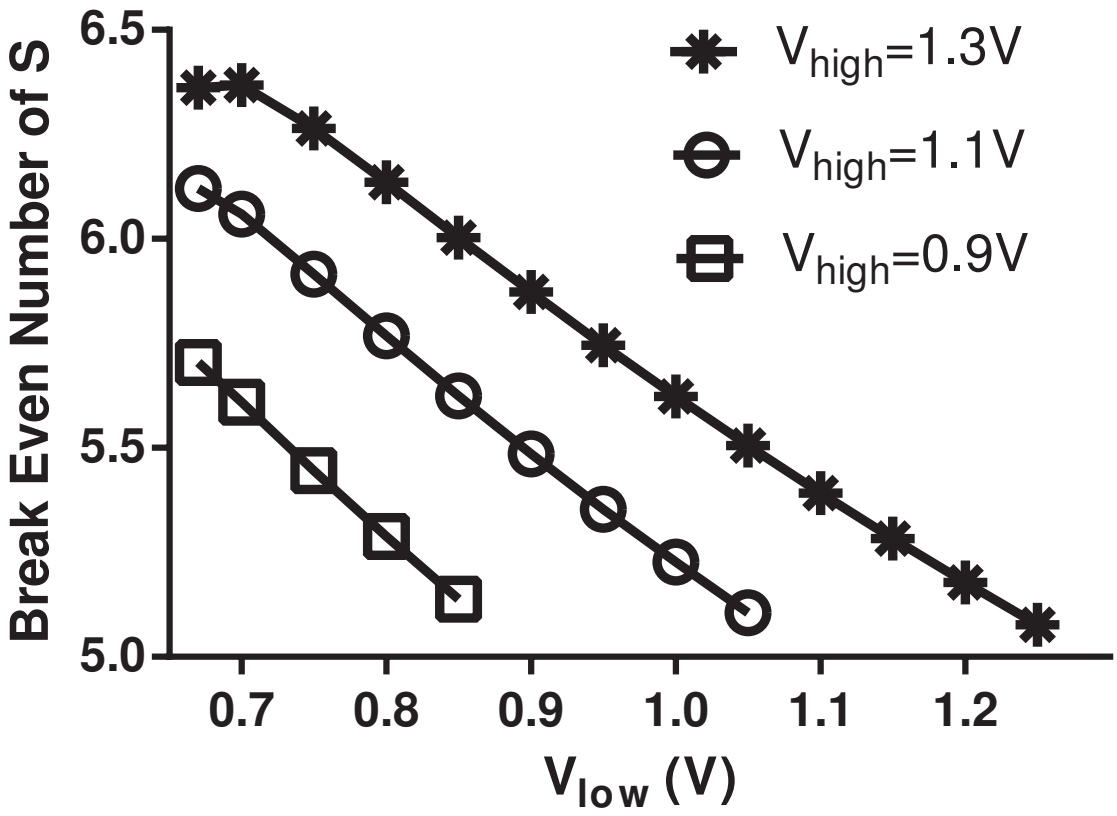
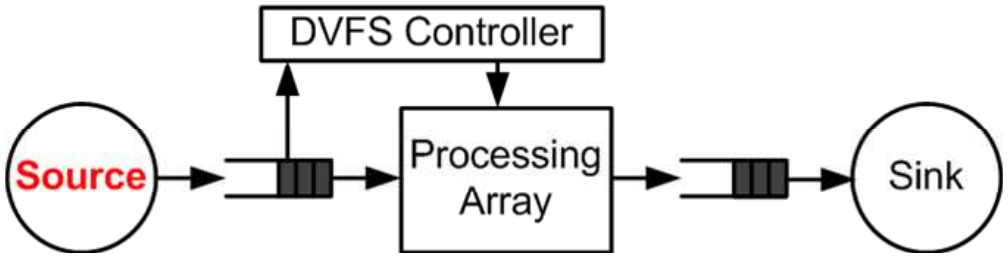
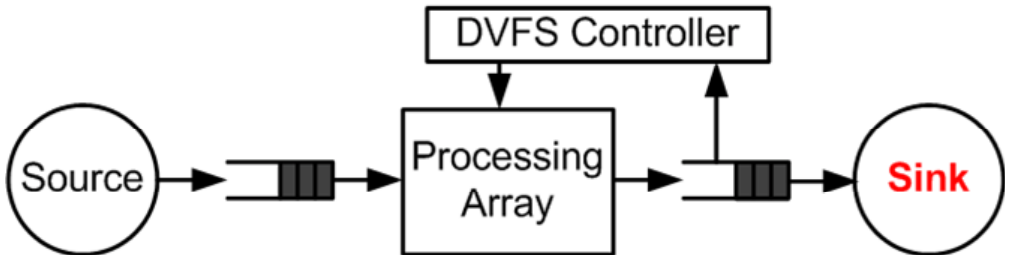


Figure 4.7: Break even number of S for E_{dith} less than E_{freq} with different V_{high}. The leakage power is measured from AsAP. $\alpha = 0.1$



(a)



(b)

Figure 4.8: DVFS controllers with constraints from (a) upstream (source) and (b) downstream (sink).

AsAP pays no penalty for reconfiguring the clock frequency, and only a few stall cycles for every voltage switch [133]. As shown in Fig. 4.6, the break even number of S for $E_{dith} < E_{freq}$ is less than seven cycles for AsAP, which implies that as long as the core executes more than seven cycles on V_{low}, the voltage switching overhead would be overcome by the extra energy saving by switching to a lower voltage supply. Fig. 4.6 also shows that the break even number of S decreases as leakage power ratio increasing. The reason is that as leakage power starts to dominate in the total power consumption, less dynamic power saving is required to break even the switching energy overhead, which results in less active cycles on V_{low}. Fig. 4.7 shows that the break even number of S also decreases, as V_{high} decreases. For many-core systems with different α, CP, T_{sw} and LR, the above analysis is necessary before determining whether frequency scaling or voltage dithering should be applied. For AsAP, voltage dithering is more energy efficient and will be applied in the following discussion of this chapter.

4.4 Proposed DVFS Local Algorithm

4.4.1 Performance Constrained Systems

In general, there are two categories of real time systems with different performance constraints, *upstream* constraint systems (UCS) and *downstream* constraint systems (DCS). The constraints from upstream are given by the source, which means that the input data has to be processed at a certain rate to ensure correct operation, like analog-to-digital conversion. On the other hand, systems could be constrained by downstream. In this case, a certain output rate is required to be satisfied for correct operation, like video processing, wireless communication and digital-to-analog conversion. For the completeness of the work, both UCS and DCS are considered for different algorithms in the rest of the chapter. Fig. 4.8 shows DVFS controllers are tuned by the inputs and outputs, for systems with upstream and downstream constraints, respectively.

4.4.2 Dithering with FIFO Occupancy

For UCS, the constraints come from the input ports of the system. When input FIFOs tend to be empty, it means that the processor runs too fast and could slow down by connecting with a lower VF pair to save energy. On the other hand, if input FIFOs are filled up, it means that the processor runs too slow and should speed up by tying with a higher VF pair to satisfy the performance requirement. For DCS, the constraints come from the output ports of the system. The processor should slow down when the output FIFOs are filled up, while speed up when the output FIFOs are empty. Two methods to adjust the VF pair based on FIFO occupancy are proposed, one is *even partition* (EP), the other is *error control* (EC).

Even Partition

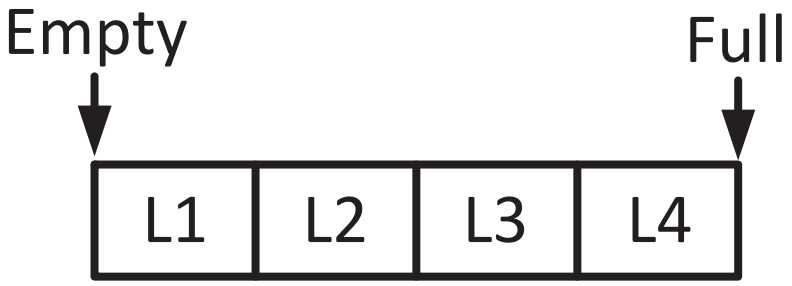
Suppose there are N discrete VF pairs set by the global algorithm (will be discussed in Section 4.5), which are (V_1, f_1), (V_2, f_2), ..., (V_N, f_N). Then each FIFO is split into $(N+1)$ levels evenly, which are $(L_1, L_2, ..., L_{N+1})$. Fig. 4.9(a) shows that the FIFO is divided into four levels with three VF pairs. For UCS, there are M input FIFO links for the core under test. The FIFO occupancy of the M input links are $(O_1, O_2, ..., O_M)$, and each of them O_i is represented by L_i. Therefore, the current FIFO occupancy is determined by $O_{curr} = MAX(O_1, O_2, ..., O_M)$. Assume

Algorithm 1 *Even Partition for UCS*

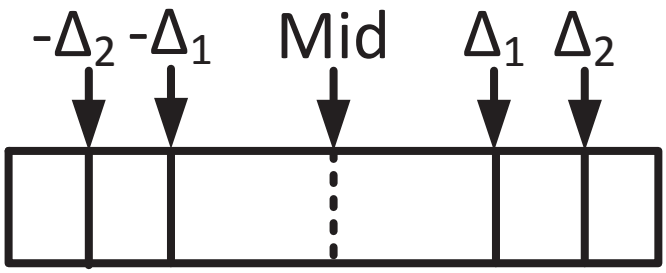
$O_{curr} = L_1$;
$(V_{next}, f_{next}) = (V_{pre}, f_{pre})$;
for $i = 1 : M$ **do**
    if $O_i > O_{curr}$ **then**
        $O_{curr} = O_i$;
    end if
end for

if $O_{curr} == L_1$ **then**
    $(V_{next}, f_{next}) = (V_1, f_1)$;
    $LastTune = down$;
else if $O_{curr} == L_{N+1}$ **then**
    $(V_{next}, f_{next}) = (V_N, f_N)$;
    $LastTune = up$;
else if $(O_{curr} > (O_{pre} +$ one level$))$ **OR**
        $(O_{curr} > O_{pre}$ **AND** $LastTune == up)$ **then**
    $(V_{next}, f_{next}) = (V_{pre}, f_{pre}) +$ one level;
    $LastTune = up$;
else if $(O_{curr} < (O_{pre} -$ one level$))$ **OR**
        $(O_{curr} < O_{pre}$ **AND** $LastTune == down)$ **then**
    $(V_{next}, f_{next}) = (V_{pre}, f_{pre}) -$ one level;
    $LastTune = down$;
end if

if $V_{pre} != V_{next}$ **then**
    $(V_{pre}, f_{pre}) = (V_{next}, f_{next})$;
    $O_{pre} = O_{curr}$;
end if



(a)



(b)

Figure 4.9: Example of FIFO partitions for (a) EP and (b) EC with three global VF pairs.

that during the previous voltage switch, the input FIFO occupancy is O_{pre} and the selected VF pair is (V_{pre}, f_{pre}). There are four basic policies used for determining (V_{next}, f_{next}) as follows

- If O_{curr} equals the **lowest** FIFO level L_1, (V_1, f_1) is assigned.
- If O_{curr} equals the **highest** FIFO level L_{N+1}, (V_N, f_N) is assigned.
- Tune up the VF pair if any of the following conditions is satisfied. (1) O_{curr} is at least two levels higher than O_{pre}; (2) O_{curr} is higher than O_{pre} **AND** the previous VF switch is **up**.
- Tune down the VF pair if any of the following conditions is satisfied. (1) O_{curr} is at least two levels lower than O_{pre}; (2) O_{curr} is lower than O_{pre} **AND** the previous VF switch is **down**.

Algorithm 1 shows the pseudocode of the even partition for UCS (EP-UCS).

For DCS, it is assumed that there are M output FIFOs from the core under test, where O_i represents the ith output FIFO occupancy. The current output FIFO occupancy can be derived as $O_{curr} = MAX(O_1, O_2, ..., O_M)$. The four basic VF pair tuning policies are listed as follows

Algorithm 2 *Even Partition for DCS*

1: $O_{curr} = L_1$;
2: $(V_{next}, f_{next}) = (V_{pre}, f_{pre})$;
3: **for** $i = 1 : M$ **do**
4: **if** $O_i > O_{curr}$ **then**
5: $O_{curr} = O_i$;
6: **end if**
7: **end for**

8: **if** $O_{curr} == L_1$ **then**
9: $(V_{next}, f_{next}) = (V_N, f_N)$;
10: $LastTune = up$;
11: **else if** $O_{curr} == L_{N+1}$ **then**
12: $(V_{next}, f_{next}) = (V_1, f_1)$;
13: $LastTune = down$;
14: **else if** $(O_{curr} > (O_{pre} +$ one level$))$ **OR**
15: $(O_{curr} > O_{pre}$ **AND** $LastTune == down)$ **then**
16: $(V_{next}, f_{next}) = (V_{pre}, f_{pre}) -$ one level;
17: $LastTune = down$;
18: **else if** $(O_{curr} < (O_{pre} -$ one level$))$ **OR**
19: $(O_{curr} < O_{pre}$ **AND** $LastTune == up)$ **then**
20: $(V_{next}, f_{next}) = (V_{pre}, f_{pre}) +$ one level;
21: $LastTune = up$;
22: **end if**

23: **if** $V_{pre} != V_{next}$ **then**
24: $(V_{pre}, f_{pre}) = (V_{next}, f_{next})$;
25: $O_{pre} = O_{curr}$;
26: **end if**

- If O_{curr} equals the **lowest** FIFO level L_1, (V_N, f_N) is assigned.
- If O_{curr} equals the **highest** FIFO level L_{N+1}, (V_1, f_1) is assigned.
- Tune down the VF pair if any of the following conditions is satisfied. (1) O_{curr} is at least two levels higher than O_{pre}; (2) O_{curr} is higher than O_{pre} **AND** the previous VF switch is **down**.
- Tune up the VF pair if any of the following conditions is satisfied. (1) O_{curr} is at least two levels lower than O_{pre}; (2) O_{curr} is lower than O_{pre} **AND** the previous VF switch is **up**.

Algorithm 2 shows the pseudocode of the even partition for DCS (EP-DCS).

Error Control

The key idea of EC is to keep the FIFO occupancy at around half of the FIFO size (S_{fifo}), which implies that the processor runs neither too fast nor too slow. When a sudden change happens on the performance requirement, the processor would have enough time to adjust its VF pair before any stall happens. Each FIFO occupancy has an offset error value compared to the targeted FIFO occupancy. For UCS, a positive offset error means more work is required to be done than expectation, and the processor should speed up to meet the performance requirement. A negative offset error means the processor could slow down for extra power saving. The current offset errors for the M input FIFOs of the core under test can be represented by $(e_1, e_2, ..., e_M)$, where $e_i = |O_i - 0.5 \times S_{fifo}|$, while $(e_{pre1}, e_{pre2}, ..., e_{preM})$ are the offset errors during the previous VF switch. Unlike EP, EC divides half of the FIFO with $(N-1)$ boundaries $(\Delta_1, \Delta_2, ..., \Delta_{N-1})$ into N uneven segments, where

$$\Delta_i - \Delta_{i-1} = 2 \times (\Delta_{i+1} - \Delta_i) \tag{4.9}$$

As shown in Fig. 4.9(b), $(\Delta_2 - \Delta_1)$ is half of $(\Delta_1 - Mid)$ for systems with three global VF pairs. The uneven partition reduces the algorithm's response delay to performance requirement adjustments. For example, if the VF pair keeps tuned up, FIFO space filled during $(V_i \rightarrow V_{i+1})$ is 50% compared with $(V_{i-1} \rightarrow V_i)$. For each input FIFO, the VF pair is determined as follows

- If $e_i > e_{prei}$ **AND** $e_i > \Delta_j \geq e_{prei}$, tune up the VF pair.
- If $e_i < e_{prei}$ **AND** $e_i < -\Delta_j \leq e_{prei}$, tune down the VF pair.

Algorithm 3 *Error Control for UCS*

$(V_{next}, f_{next}) = (V_1, f_1)$;
for $i = 1 : M$ **do**
    $e_i = O_i - 0.5 \times S_{fifo}$;

    if $e_i > e_{prei}$ **AND** $e_i > \Delta_j \geq e_{prei}$ **then**
        $(V_{temp}, f_{temp}) = (V_{pre}, f_{pre})$ + one level;
    else if $e_i < e_{prei}$ **AND** $e_i < -\Delta_j \leq e_{prei}$ **then**
        $(V_{temp}, f_{temp}) = (V_{pre}, f_{pre})$ − one level;
    else
        $(V_{temp}, f_{temp}) = (V_{pre}, f_{pre})$;
    end if

    if $V_{temp} < V_{next}$ **then**
        $(V_{next}, f_{next}) = (V_{temp}, f_{temp})$;
    end if
end for

if $V_{pre} != V_{next}$ **then**
    $(V_{pre}, f_{pre}) = (V_{next}, f_{next})$;
    $(e_{pre1}, e_{pre2}, ..., e_{preM}) = (e_1, e_2, ..., e_M)$;
end if

Then the highest VF pair is applied as (V_{next}, f_{next}). Algorithm 3 shows the pseudocode of the error control for UCS (EC-UCS).

Algorithm 4 *Error Control for DCS*

$(V_{next}, f_{next}) = (V_N, f_N)$;
for $i = 1 : M$ **do**
    $e_i = O_i - 0.5 \times S_{fifo}$;

    if $e_i > e_{prei}$ **AND** $e_i > \Delta_j \geq e_{prei}$ **then**
        $(V_{temp}, f_{temp}) = (V_{pre}, f_{pre})$ − one level;
    else if $e_i < e_{prei}$ **AND** $e_i < -\Delta_j \leq e_{prei}$ **then**
        $(V_{temp}, f_{temp}) = (V_{pre}, f_{pre})$ + one level;
    else
        $(V_{temp}, f_{temp}) = (V_{pre}, f_{pre})$;
    end if

    if $V_{temp} < V_{next}$ **then**
        $(V_{next}, f_{next}) = (V_{temp}, f_{temp})$;
    end if
end for

if $V_{pre} != V_{next}$ **then**
    $(V_{pre}, f_{pre}) = (V_{next}, f_{next})$;
    $(e_{pre1}, e_{pre2}, ..., e_{preM}) = (e_1, e_2, ..., e_M)$;
end if

On the other hand, a positive offset error in the output FIFO of DCS means the processor could slow down, while a negative offset error implies the processor should speed up. As a result, for each output FIFO, the VF pair is determined as follows

- If $e_i > e_{prei}$ **AND** $e_i > \Delta_j \geq e_{prei}$, tune down the VF pair.

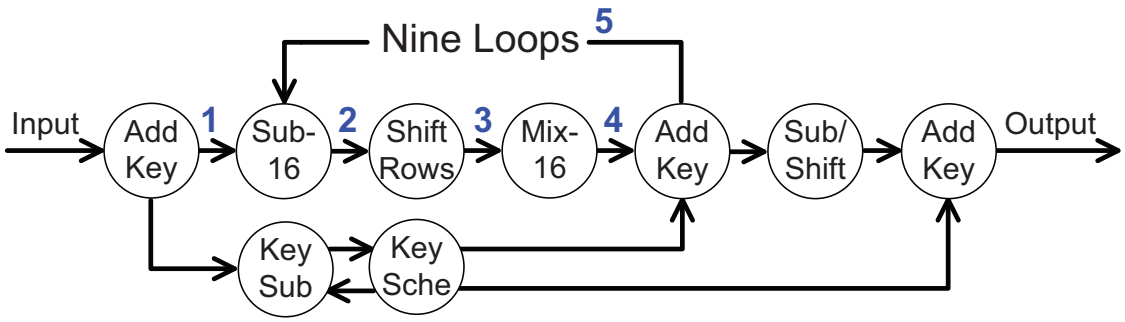


Figure 4.10: 9-core AES engine dataflow diagram.

- If $e_i < e_{prei}$ **AND** $e_i < -\Delta_j \leq e_{prei}$, tune up the VF pair.

Then the lowest VF pair is selected as (V_{next}, f_{next}). Algorithm 4 shows the pseudocode of the error control for DCS (EC-DCS).

4.4.3 Dithering with FIFO Stall Information

Workload Inheritance for UCS

Although FIFO occupancy is effective to help processors choose appropriate VF pairs, it provides wrong decisions in some scenarios. Fig. 4.10 shows the dataflow diagram of a 9-core Advanced Encryption Standard (AES) engine [40]. Considering the AES engine has an upstream constraint, the VF pairs adjust based on input FIFO occupancy. When the performance requirement increases, the input FIFO for the whole engine is filled up, and speeds up *AddKey* at the front of the dataflow. Then, FIFO 1 is tend to be full, and *Sub-16* in the loop is assigned to a higher VF pair. In the AES engine, only one 16-byte data block can be processed in the loop at anytime. Therefore, FIFO 2, 3, and 4 would never be filled up enough to raise the VF pair assignments of *ShiftRows*, *Mix-16* and *AddKey* in the loop. As a result, the AES engine fails to meet the desired throughput requirement. From the above example, it shows that selecting VF pair only based on FIFO occupancy for UCS may cause failures.

In order to solve the problem, we propose a new technique called *workload inheritance* (WI), which is based on FIFO stall information. Assume that one of the *coreA* input FIFOs is connected with one of the *coreB*'s outputs. If *coreA* is stalled on *coreB* due to empty on input

(EOI), and *coreA* is not tied with the lowest VF pair, then *coreB* is considered to inherit workload from *coreA*. Each core has a *stall counter*, C_{stall}, which counts up if it inherits workload from any of its output cores; otherwise counts down. For systems with N global VF pairs, there are $(N-1)$ stall counter thresholds $(T_1, T_2, ..., T_{N-1})$. The VF pair of the core under test is determined as

$$(V_{next}, f_{next}) = \begin{cases} (V_1, f_1), & \text{if } C_{Stall} \leq T_1 \\ (V_i, f_i), & \text{if } T_{i-1} < C_{Stall} \leq T_i \\ (V_N, f_N), & \text{if } C_{Stall} > T_{N-1} \end{cases} \tag{4.10}$$

Suppose the core under test has P output FIFOs, the workload inheritance algorithm is shown in Algorithm 5.

Considering the processors in the loop of the above example, when *Sub16* speeds up due to FIFO 1, while *ShiftRows*, *Mix16*, and *AddKey* run at lower frequencies than required, *Sub16* would be eventually stalled on *AddKey* due to EOI on FIFO 5. Therefore, *AddKey* in the loop starts to inherit workload from *Sub16*, and speed up. Similarly, the VF pairs of *ShiftRows* and *Mix16* would be also tuned up until the performance requirement is satisfied.

Workload Inheritance for DCS

For DCS, the processors select their VF pairs based on output FIFO occupancy. When the performance requirement decreases, *Sub-16*, *ShiftRows*, and *Mix-16* in the loop would not be able to slow down since FIFO 2, 3 and 4 cannot be filled. As a result, selecting VF pairs only based on FIFO occupancy for DCS may miss power saving opportunities and cause energy inefficiency.

A similar WI scheme is applied to solve the above problem. Assume that one of *coreA* output FIFOs is connected with one of *coreB*'s inputs. If *coreB* is stalled on *coreA* due to EOI, and *coreA* is not tied with the highest VF pair, then *coreB* is considered to inherit workload from *coreA*. The VF pair of the core under test is determined as

$$(V_{next}, f_{next}) = \begin{cases} (V_N, f_N), & \text{if } C_{Stall} \leq T_1 \\ (V_\alpha, f_\alpha), & \text{if } T_{i-1} < C_{Stall} \leq T_i \\ (V_1, f_1), & \text{if } C_{Stall} > T_{N-1} \end{cases} \tag{4.11}$$

Algorithm 5 *Workload Inheritance for UCS*

$C_{inc} = 0$
for $i = 1 : P$ **do**
    if $Core_i$ is stalled **AND** V_{corei} != V_1 **then**
        for $j = N : 2$ **do**
            if $V_{corei} == V_j$ **then**
                $C_{inc} = \max(C_{inc},\ j - 1)$;
                break;
            end if
        end for
    end if
end for

if C_{inc} != 0 **then**
    C_{Stall} += C_{inc};
else
    C_{Stall} -= 1;
end if

$(V_{next}, f_{next}) = (V_1, f_1)$;
for $i = N : 2$ **do**
    if $C_{Stall} > T_{i-1}$ **then**
        $(V_{next}, f_{next}) = (V_i, f_i)$
        break;
    end if
end for

where $\alpha = N - i + 1$. Suppose the core under test has P input FIFOs, the WI algorithm for DCS (WI-DCS) is shown in Algorithm 6.

Considering the loop in Fig. 4.10, when *AddKey* slows down due to full of its output FIFO, while *Sub-16*, *ShiftRows*, and *Mix-16* run at higher frequencies than required, *Sub-16* would be eventually stalled on *AddKey* due to EOI on FIFO 5. Therefore, *Sub-16* in the loop starts to inherit workload from *AddKey*, and slows down. Similarly, the VF pairs of *ShiftRows* and *Mix-16* would be also tuned down for extra power saving.

By combining the proposed algorithms based on FIFO occupancy and stall information, we get two different local algorithms for both UCS and DCS, respectively: one is EP+WI, the other is EC+WI. For each local algorithm, the VF selection schemes based on FIFO occupancy and stall information are executed in orthogonal. Then, the higher VF pair from the results is assigned to the core under test to guarantee performance requirements for UCS, while the lower VF pair is assigned for DCS to maximize power saving. The two local algorithms will be compared in detail in terms of power saving, VF switching frequency and response delay to workload variation in Section 4.6.

4.5 Proposed DVFS Global Algorithm

The local DVFS algorithms help individual processor select the most energy efficient VF pair according its workload, while the global DVFS algorithm is adopted to pick global VF pairs for the whole chip based on all active processors workload. The motivation example is shown in Fig. 4.11. Considering the targeted system has two global VF pairs, (V_{low}, V_{high}) is selected at T_1 as the optimal global voltage setting based on the processors workload distribution $(d_1, d_2, ..., d_n)$, which is marked by the dots. The total power consumption is derived as follows

$$P = \sum_{i=1}^{n} P_j = \sum_{i=1}^{n} H(d_i) \tag{4.12}$$

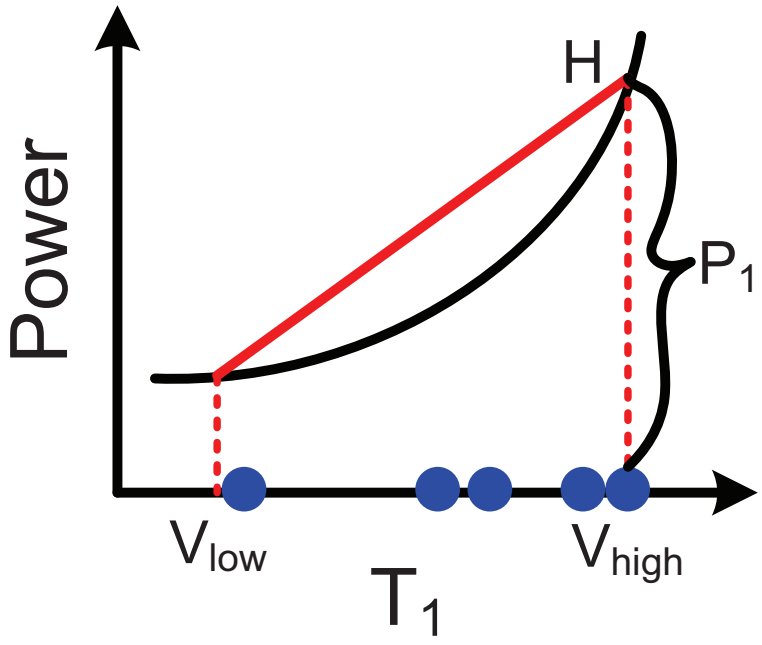
where H is the function of the straight line that connects V_{low} and V_{high} on the power curve. As shown in Fig. 4.11(a), $P_1 = H(d_1)$ is the power consumption of the processor with the largest workload at T_1. However, the processors' workload distribution for a many-core system is unstable. Either a varied performance requirement or an additional task mapping on the system is capable of shifting the workload distribution. As shown in Fig. 4.11(b), most processors have a lighter

Algorithm 6 *Workload Inheritance for DCS*

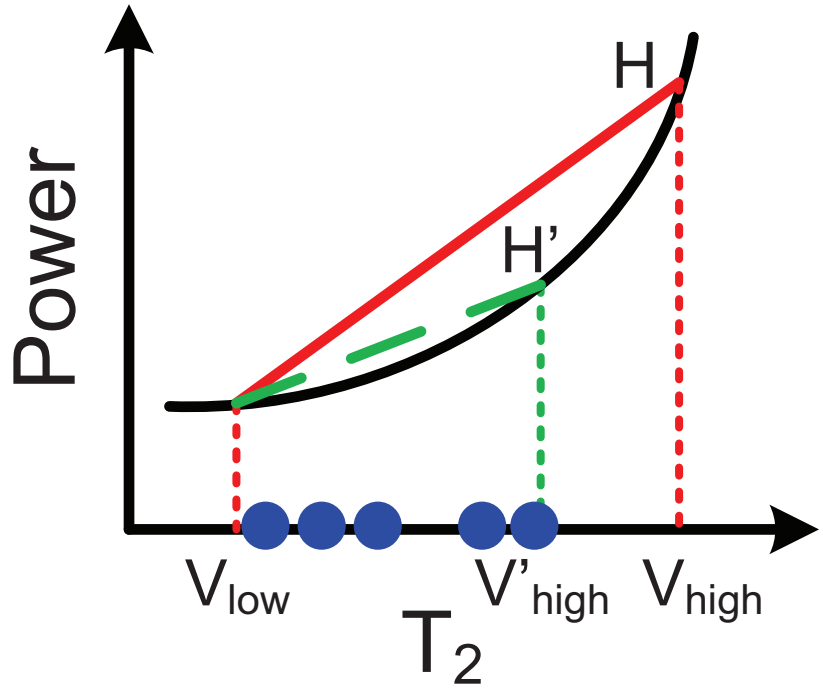
$C_{inc} = 0$
for $i = 1 : P$ **do**
  if stalled on $FIFO_i$ **AND** $V_{corei} != V_N$ **then**
    for $j = 1 : N - 1$ **do**
      if $V_{corei} == V_j$ **then**
        $C_{inc} = \max(C_{inc},\ N - j)$;
        break;
      end if
    end for
  end if
end for

if $C_{inc} != 0$ **then**
  $C_{Stall} += C_{inc}$;
else
  $C_{Stall} -= 1$;
end if

$(V_{next}, f_{next}) = (V_N, f_N)$;
for $i = N : 2$ **do**
  if $C_{Stall} > T_{i-1}$ **then**
    $(V_{next}, f_{next}) = (V_{N-i+1}, f_{N-i+1})$
    break;
  end if
end for



(a)



(b)

Figure 4.11: Power consumption versus the global voltage/frequency pair settings. The dots illustrate that the processors workload distribution is changing over time.

workload at T_2 compared to T_1. By decreasing V_{high} to V'_{high}, the power function is changed from H to H', which significantly reduces the total power consumption. Therefore, to fully extract the power saving potential of many-core systems, it requires not only local DVFS algorithms for choosing the best VF pair for each individual core, but also global algorithms to tune the global VF pairs for the whole chip based on the overall workload distribution.

To provide accurate workload information to the global controller, each core is required to record both the number of stall cycles Ns_i and active cycles Na_i when it is assigned to the VF pair (V_i, f_i). Then the desired operation frequency can be derived as

$$f_{aim} = \begin{cases} \frac{Na_1}{(Ns_1+Na_1)/f_1}, & \text{only tied } (V_1, f_1) \\ \\ \frac{\sum_{i=1}^{N}(Ns_i+Na_i)}{\sum_{i=1}^{N}((Ns_i+Na_i)/f_i)}, & \text{otherwise} \end{cases} \tag{4.13}$$

And the active percentage is

$$Act = \frac{\sum_{i=1}^{N}(Na_i/fi)}{\sum_{i=1}^{N}((Ns_i + Na_i)/f_i)} \tag{4.14}$$

Based on the desired frequency and active percentage of all the cores, the global controller can loop over all possible VF pair combinations and find the most energy efficient VF setting for the chip. The brute force method usually works well since both the number of global VF pairs and the number of available VF levels are relative small. Additionally, the frequency of global VF optimization should be relatively low due to the large overhead brought by tuning voltage regulators. The pseudocode of the global VF optimization is shown in Algorithm 7. A special case is considered at the beginning of the algorithm. For UCS, when the input FIFOs of the system tend to be full (greater than a preset threshold), it means that the current global VF pair setting is not able to satisfy the performance requirement. As a result, a default VF setting is applied instantly. On the other hand, the default VF setting is also applied when the output FIFOs of the system tend to be empty (smaller than a pre-selected threshold) for DCS.

4.6 Experimental Results

First, we present the benchmarks that are used in the simulation. Secondly, the two proposed local algorithms, EP+WI and EC+WI, are studied in detail with a 9-core AES engine.

Algorithm 7 *Global VF Optimization*

P_{curr} = Power when all cores are assigned with the largest available VF pair;
if system input FIFOs Occup. $> Th$ for UCS **OR** system output FIFOs Occup. $< Th$ for DCS **then**
    return $VF_{opt} = VF_{default}$;
end if

for each of all possible VF pair combinations VF_{curr}: $((V_1, f_1), ..., (V_N, f_N))$ **do**
    $P_{temp} = 0$;
    for each active core j **do**
        find $f_i \geq f_{aimj} > f_{i-1}$;
        $Per_i = (f_{aimj} - f_{i-1})/(f_i - f_{i-1})$;
        $P_{temp} +=$ EstimatePower(Per_i, V_i, Act_j);
    end for
    if $P_{curr} > P_{temp}$ **then**
        $P_{curr} = P_{temp}$;
        $VF_{opt} = VF_{curr}$;
    end if
end for
return VF_{opt};

Then, the two local algorithms are compared in terms of power saving, VF switching frequency and the response delay to workload variations. Finally, the extra power saving brought by the global algorithm is investigated for systems with different performance constraints, number of global VF pairs and voltage regulator tuning resolution.

4.6.1 Benchmarks

Most of dataflows could be represented by a combination of the six basic dataflow patterns shown in Fig. 4.12. To cover all of the basic dataflow patterns and many of their combinations, the benchmark includes an implementation library with seven AES engines [40], a JEPG encoder [134],

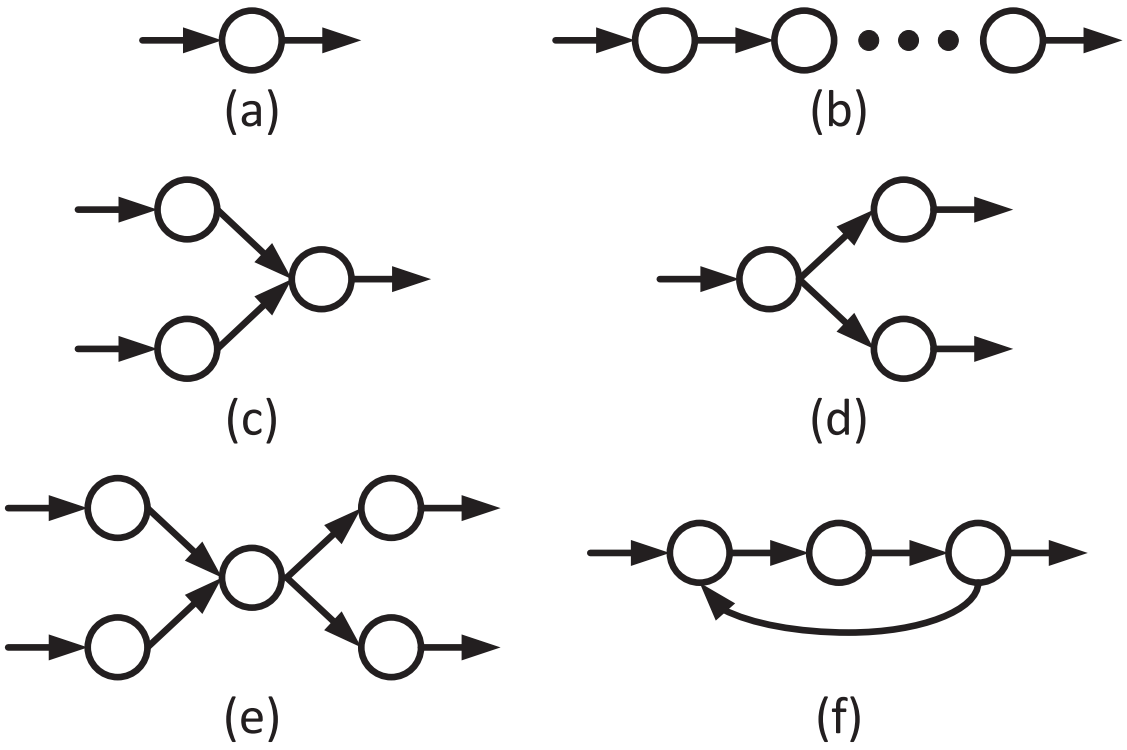


Figure 4.12: Basic dataflow patterns: (a) single-input single-output; (b) chain; (c) multiple-input single-output; (d) single-input multiple-output; (e) multiple-input multiple-output; (f) loop.

Table 4.1: Number of cores and throughput of different benchmarks

Benchmarks	Cores Usage	1/Throughput (cycles/byte)
AES Small	8	167.375
AES One-task one-processor	9	223.875
AES Parallel-MixColumns	15	136.250
AES Parallel-SubBytes-MixColumns	18	84.375
AES Loop-unrolled Three Times	23	68.625
AES Loop-unrolled Nine Times	50	16.625
AES No-merge-parallelism	59	9.500
JPEG Encoder[a]	9	21.875
802.11a Transmitter[a]	22	148

[a] The throughput of JPEG encoder and 802.11a transmitter are data dependent.

[b] The benchmarks are labeled from 1 to 9 in the following sections for simplicity.

Table 4.2: Optimal frequencies (MHz) and working frequencies selected by the proposed DVFS algorithms of processors in the 9-core AES engine.

Processor		UCS		DCS	
Name	Optimal	EP+WI	EC+WI	EP+WI	EC+WI
AddKey_Head	15	15	15	15	15
SubByte-4	1210	1210	1210	1210	1210
MixColumn-4	1210	1210	1210	1210	1210
ShiftRows	1210	1210	1210	1210	1210
AddKey	1210	1210	1210	1210	1210
KeySub	277	360	360	291	294
KeySche	277	296	329	291	294
Sub/Shift	50	50	50	50	50
AddKey_Tail	15	15	15	15	15

and a 802.11a baseband transmitter [30]. Table 4.1 lists the number of cores and throughput for different benchmarks. As shown in Fig. 4.10, even a simple 9-core AES engine covers all of the basic dataflow patterns.

4.6.2 Case Study: 9-core AES Engine

As discussed in Section 4.4, two local DVFS algorithms are proposed, which are EP+WI and EC+WI, for both UCS and DCS, respectively. To demonstrate that the proposed local algorithms are capable of selecting VF pairs properly, the One-task one-processor (OTOP) 9-core AES engine is studied in detail. Without loss of generality, we assume that there are three global VF pairs (V, MHz) for the whole chip, which are set as (1.3, 1210), (1.09, 895) and (0.87, 498). The VF pairs are selected according to a static optimization by assuming that the workload of all available cores on the targeted platform is evenly distributed from 0% to 100%. Also, the performance requirement of the OTOP AES engine is set to 43 Mbps, which is the maximum throughput allowed.

Table 4.2 shows the comparison between the optimal frequencies and the working frequencies selected by the proposed algorithms. The optimal frequency of each core is obtained with static

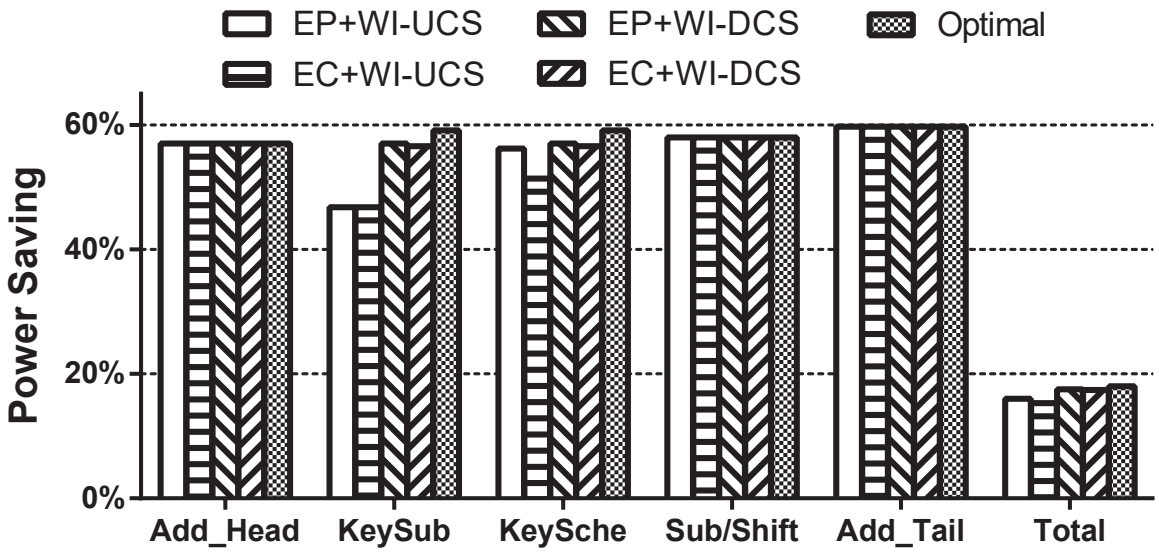


Figure 4.13: Power saving of individual cores in the 9-core AES engine.

workload analysis, while the working frequency selected by DVFS algorithms is defined by

$$f_{DVFS} = \sum Per_i \cdot f_i \tag{4.15}$$

where Per_i is the percentage of time the core spends on (V_i, f_i).

As shown in Table 4.2, all the four cores in the loop are tied with the highest VF pair all the time, while other cores have great potential for power saving. Fig. 4.13 shows the power saving for each individual core from three different solutions, including EP+WI-UCS, EC+WI-UCS, EP+WI-DCS, EC+WI-DCS and the optimal static workload analysis. Both EP+WI and EC+WI can save as much as 60% power for cores that are not on the critical path. Overall, EP+WI shows 16% and 17% power saving under the throughput constraint for UCS and DCS, which is only 2% and 1% less than the optimal solution. EC+WI shows 15% and 17% power saving for UCS and DCS, respectively.

4.6.3 Local Algorithm Comparison

To compare the performance of the proposed local algorithms, three metrics are used as follows

1. power saving;

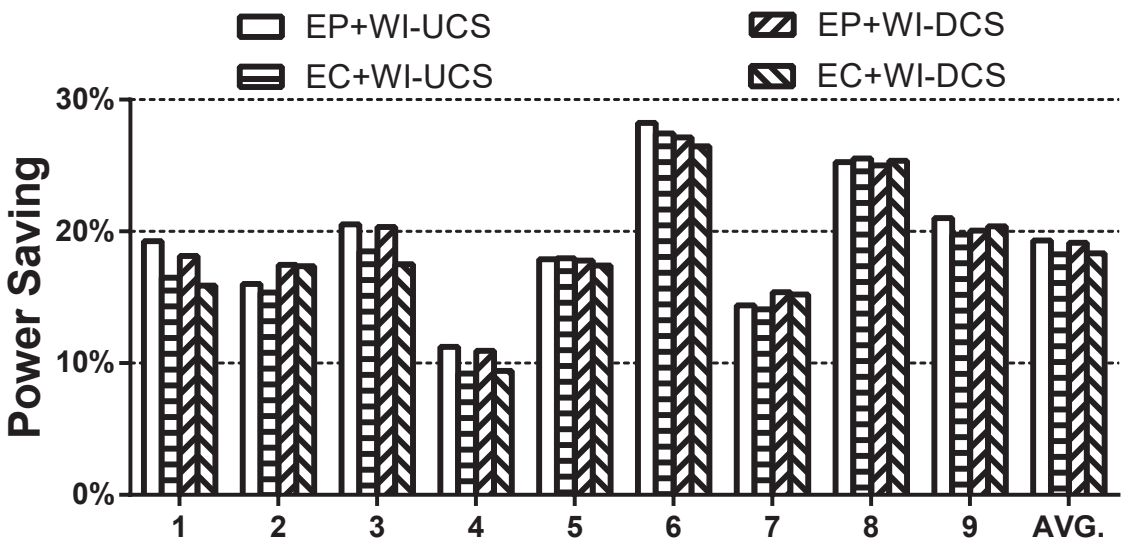


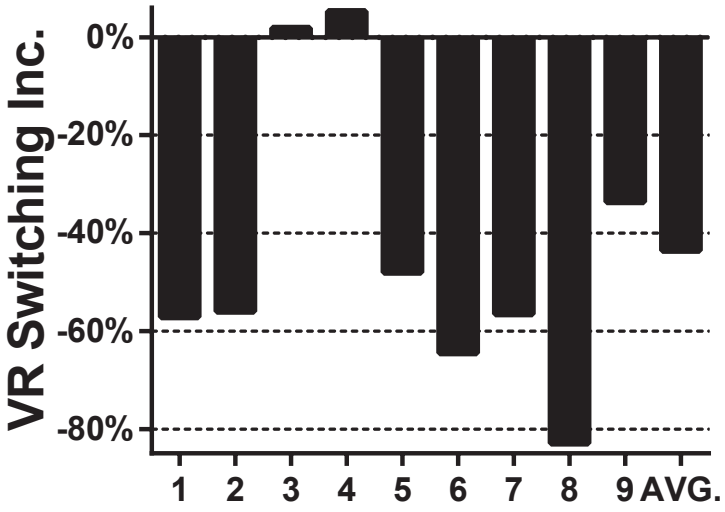
Figure 4.14: Power saving of the proposed local algorithms for benchmarks in Table 4.1.

2. frequency of VF pair switching;

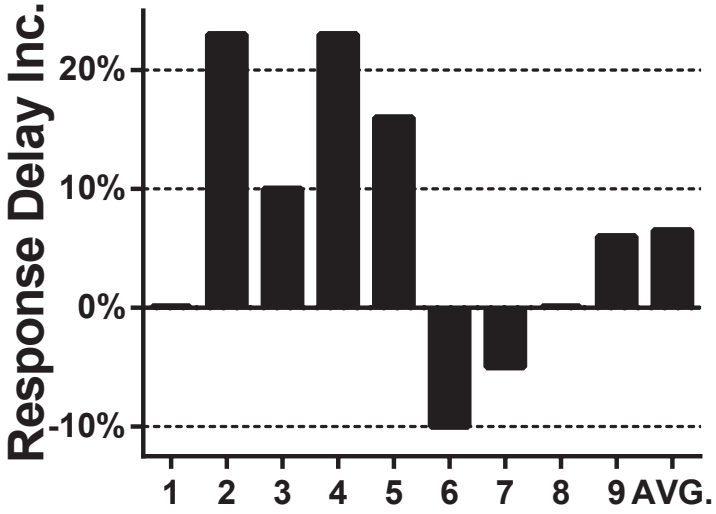
3. response delay to workload change.

Power saving is the essential metric to evaluate the effectiveness of an DVFS algorithm. Additionally, a large number of VF pair switchings are not desirable since they are both energy and time expensive as discussed in Section 4.3. Furthermore, a quick response to workload variation not only reduces the opportunity of missing deadlines when the performance requirement increases, but also increases the time window for power saving when the performance requirement decreases.

In this subsection, we assume that the three global VF pairs are set statically without considering the global algorithm as discussed in Section 4.6.2. The power saving and VF switching frequency are measured when benchmarks runs with maximum throughput. The response delay is measured by changing the performance requirement from 10% to 90% of the maximum throughput. Also, only the UCS are considered in this section for simplicity. As shown in Fig. 4.14, EP+WI shows a slightly better performance in terms of power saving for most of the test cases. In average, EP+WI saves 19.3% power, which is 1.1% more than the 18.2% brought by EC+WI. Fig. 4.15(a) shows the normalized incremental VF switching frequency of EC+WI compared with EP+WI. Although the VF switching frequency highly depends on the dataflow, EC+WI requires 40% to



(a) Incremental VF Switching



(b) Incremental Response Delay

Figure 4.15: Comparison of EP+WI and EC+WI. (a) Normalized incremental VF switching frequency and (b) Normalized incremental response delay to performance requirement changes of EC+WI compared to EP+WI.

Table 4.3: Summary of Local Algorithms Performance Evaluation

Algorithm	Power Saving	VR Switching Frequency	Response Delay
EP+WI	+	-	+
EC+WI	+	+	-

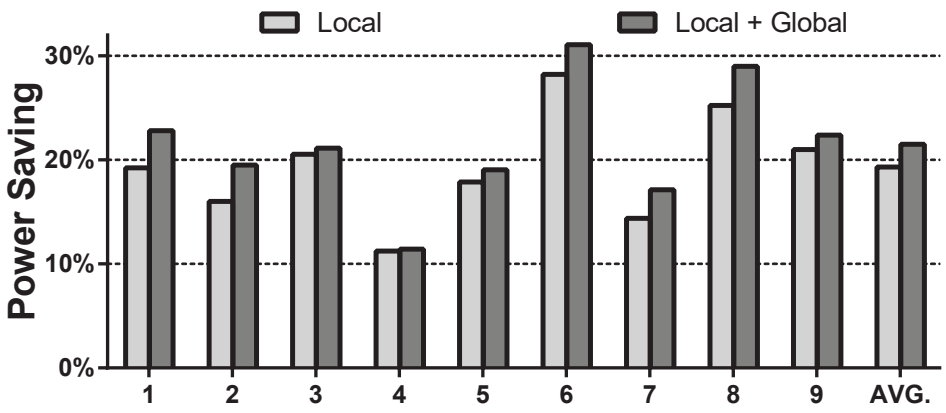
80% less VF switching than EP+WI for most of the benchmarks, which leads to a 44% VF switching frequency reduction in average. On the other hand, EC+WI requires 7% more response time than EP+WI to performance requirement changes as shown in Fig. 4.15(b).

The fundamental reason behind the results is that EP+WI is more sensitive to workload variation, which results in more frequent VF switching, but also provides more fine-tuning opportunities for each individual core to save extra power. Additionally, the sensitiveness also leads to a faster response to any workload change on system level. Table 4.3 shows the summary of the proposed local algorithms performance. EP+WI and EC+WI both have their advantages and disadvantages. For different many-core platforms and system requirements, a careful evaluation is desired before any algorithm is applied.

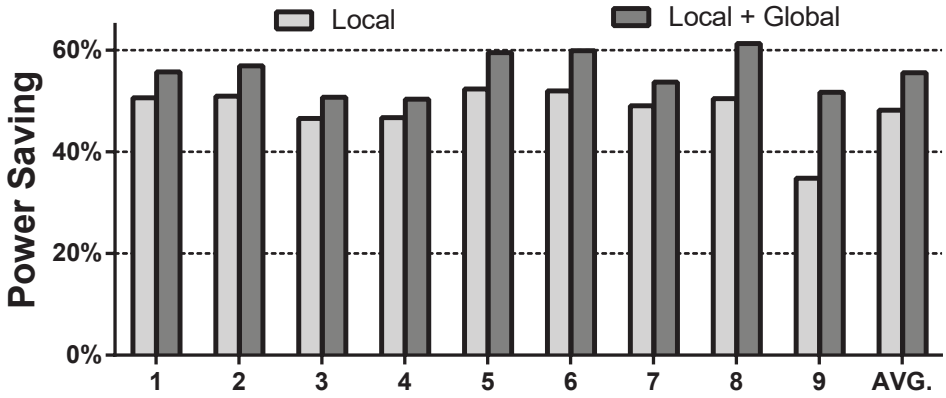
4.6.4 Global Algorithm Evaluation

Suppose three global VF pairs are available for the targeted many-core system. As discussed in Section 4.6.3, the VF pairs (V, MHz) are set as (1.3, 1210), (1.09, 895) and (0.87, 498) when global optimization is unavailable. To evaluate the performance of the proposed global algorithm, the power saving for different benchmarks are compared between local EP+WI with and without global optimization. Additionally, three different levels of performance constraint are considered, including (1) tight (= TH_{max}), (2) intermediate (= 50% TH_{max}) and (3) loose (= 10% TH_{max}), where TH_{max} is the maximum throughput of different benchmarks. The tuning resolution of off-chip voltage regulators is assumed as fine as 0.01 V.

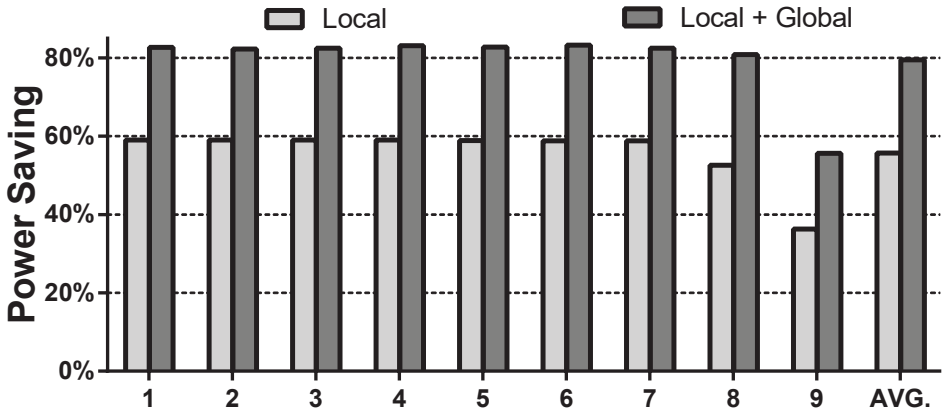
As shown in Fig. 4.16, the extra power saving brought by the global optimization is highly correlated with the performance requirement. With a tight performance constraint, the global optimization can bring average 2% extra power saving. As the performance constraint goes toward intermediate and loose, the average extra power saving increases to 7.4% and 23.8%, respectively.



(a) Tight Performance Constraint



(b) Intermediate Performance Constraint



(c) Loose Performance Constraint

Figure 4.16: Power saving comparison between local with and without global optimization for different performance constraints (a) Tight (b) Intermediate (c) Loose.

The reason is that the lower the performance constraint, the larger the offset between the optimal and default VF settings. As a result, there is more extra power saving can be achieved by global optimization. For example, the optimal VF pairs for OTOP with a tight throughput requirement is (1.3, 1210), (1.13, 958) and (0.77, 296), which is much closer to the default VF setting compared with the optimal VF pairs (0.7, 146), (0.69, 124) and (0.67, 80) for a loose throughput requirement.

Different Number of Global VF Pairs

In this subsection, we analyze the impact of the number of VF pairs on the global optimization. There are six settings are studied:

- Two global VF pairs with local (Two-L), and local plus global optimization (Two-LG).
- Two global VF pairs with local (Three-L), and local plus global optimization (Three-LG).
- Two global VF pairs with local (Fout-L), and local plus global optimization (Four-LG).

Table 4.4 lists the average power saving in percentage of the proposed algorithms with and without global optimization for systems with different number of global VF pairs. With two global VF pairs, the extra power saving brought by the global optimization is 3.0%, 7.9% and 34.8% for a tight, intermediate, and loose performance constraint, respectively. However, extra power saving is reduced to 1.9%, 7.0% and 17.7% when the number of global VF pairs is increased to four. Therefore, the global optimization is more effective when there are less global VF pairs. Additionally, Three-LG is capable of saving more power than Four-L by 0.8%, 5.9% and 16.4% for different performance constraints. Therefore, optimizing the global VF pairs dynamically is more effective than increasing the number of global voltage supplies. Table 4.4 also shows that the power saving from adding extra power supply is diminished as the number of VF pair increases. For example, the average power saving of benchmarks with tight performance requirements is improved by 2.7% when the number of voltage supplies is increased from two to three, while only 1.3% from three to four.

Different Tuning Resolution of VR

Table 4.5 shows that by increasing the tuning resolution of voltage regulators from 0.01 V to 0.05 V, the extra power saving brought by global optimization are reduced to 1.8%, 6.0% and

Table 4.4: Average power savings in percentage of local with and without global optimization for systems with different number of VF pairs.

Number of Global VF Pairs	Tight Perf. Target			Inter. Perf. Target			Loose Perf. Target		
	Local	Local + Global	Extra Saving	Local	Local + Global	Extra Saving	Local	Local + Global	Extra Saving
Two	14.6%	17.6%	3.0%	42.2%	50.1%	7.9%	42.0%	76.8%	34.8%
Three	19.1%	21.6%	2.5%	48.1%	55.6%	7.5%	55.8%	79.5%	23.7%
Four	20.8%	22.7%	1.9%	49.7%	56.7%	7.0%	63.1%	80.8%	17.7%

Table 4.5: Average Extra Power Saving brought by global optimization with tuning resolution as 0.05 V and 0.01 V for systems with three VF pairs, reqpectively.

Resolution	Tight Perf. Target	Inter. Perf. Target	Loose Perf. Target
0.01 V	2.5%	7.5%	23.7%
0.05 V	1.8%	6.0%	22.6%

22.6% for tight, intermediate, and loose performance constraints, respectively.

4.7 Conclusion

In this chapter, we have presented a joint local and global DVFS scheme for many-core systems with limited number of voltage supplies. The local algorithm is used to select the most energy efficient voltage and frequency pair for each individual core based on its workload, while the global algorithm is used to choose the global voltage frequency settings for the whole chip based on all active processors workload. Two local algorithms are proposed based on FIFO occupancy and stall information for both upstream and downstream constrained systems. And their performance are evaluated in terms of power saving, voltage switching frequency and response delay to workload variations with real application benchmarks. Compared with only applying local algorithm, the global optimization is capable of reducing power further by 2% to 35% for different performance constraints. Either increasing the number of the global voltage supplies or the tuning step of off-chip voltage regulators reduces the effectiveness of the global optimization.

Chapter 5

Optimizing Power of Many-Core Systems by Exploiting Dynamic Voltage, Frequency and Core Scaling

5.1 Introduction

For the past half century, Moore's Law has been the fundamental driver of high-performance computing. The continued CMOS technology scaling doubles the transistor density of VLSI systems and had provided a predictable 40% performance improvement of single-core processors for every 18 to 24 months [135]. However, as Dennard Scaling ends, which means that the threshold voltage of transistors stops scaling along with their lithographic dimensions, the era of scaling frequency and performance without increasing power density is over. Since 2005, the semiconductor industry shifted to multi-core and many-core processors in order to sustain the proportional scaling of performance along with transistor count increases. Many-core processors with network-on-chip interconnects have been demonstrated as promising architectures for high-performance energy-efficient computing [91], [24]. As the technology scales, a single chip with 1000+ processors was recently reported [29], [123].

One of the critical challenges for many-core system design is to reduce the power dissipation and improve the energy efficiency of the chip. it is predicted that without introducing novel

Table 5.1: Comparison of Various Low Power Design Techniques

Power Reduction Techniques	Dynamic Power Saving	Leakage Power Saving
Clock Gating	Yes	No
Power Gating	No	Yes
Multi-V_{th} [136]	No	Yes
Adaptive Body Biasing [64]	No	Yes
DVFS [125], [71], [129], [73], [74]	Yes	Minor [a]
Chip-Wide DVFCS [82], [83]	Yes	Yes
Per-Core DVFCS (This Work)	Yes	Yes

[a] DVFS aims to reduce dynamic power by scaling down voltage and frequency; however, scaling down voltage can also reduce leakage power.

low power architectures or techniques, more than 50% of the chip has to be turned off due to power concerns when CMOS technology shrinks to 8 nm [21]. Researchers are eager to seek innovative solutions to relieve the "dark silicon" problem and effectively convert transistors to performance. Additionally, processors with high energy efficiency not only save millions of dollars in energy bill for supercomputers and data centers, but also extend the battery life for mobile devices.

Table 5.1 lists several low power techniques have been proposed and adopted to reduce dynamic power, or leakage power, or both. The most commonly used technique to reduce dynamic power is clock gating, which reduces the overall effective capacitance by disabling the inactive portions of the clock tree. Power gating is a widely used technique to reduce leakage power. It uses sleep transistors to shut-off the idle function blocks or even the entire processor from power supply.

Multi-V_{th} uses transistor libraries with multiple threshold voltages (V_{th}) on the same die. Transistors with higher V_{th} have higher performance and leakage power as well. On the other hand, transistors with lower V_{th} have a larger delay but less leakage power dissipation. The logic gates on non-critical paths can be assigned as high V_{th} during design time to reduce leakage power, as long as there is no performance penalty [136]. Instead of selecting V_{th}s statically, adaptive body biasing (ABB) is capable of tuning the V_{th}s of gates during runtime, by controlling the transistor

body-source voltage. A reverse body increases V_{th}, thus reducing leakage power at the cost of slowing the devices. Alternatively, a forward body bias increases speed while increasing leakage power. ABB can also used to alleviate the impact of process, voltage, and temperature (PVT) variations [64].

Dynamic voltage and frequency scaling (DVFS) exploits the fact that dynamic power is proportional to $V^2 \times freq$, to perform dynamic voltage and frequency scaling in order to provide "just-enough" processor speed to finish the workload under time/performance constraints, while reducing dynamic power dissipation at meantime. Additionally, scaling down voltage can also reduce leakage power. Many-core processors with per-core DVFS are capable of reducing energy dissipation significantly by adapting each core's supply voltage and working frequency according to its workload [125]. Various DVFS schemes for many-core processors have been discussed in the literature. Wu *et al.* formally described a nonlinear model for the FIFO occupancy, and presented a proportional-integral-derivative (PID) controller, which requires detailed analysis of the FIFO behavior before actual hardware implementation [71]. Orgas *et al.* presented an adaptive feedback controller based on state-space models to determine the optimal voltage frequency islands (VFI) for different cores [129]. Choudhary and Marculescu approached the optimal VFI by counting the stall time from both the producer and the consumer of a communication link periodically [73]. Liu *et al.* proposed a scalable on-line hardware based DVFS scheme based on both FIFO occupancy and stall information between communication links. The proposed method not only takes advantage of the fast voltage/frequency response to workload variation by monitoring FIFO occupancy, but also utilize stall information to cover the scenarios that are not solvable by FIFO occupancy [74].

To further improve the energy efficiency and performance for many-core processors, combination of DVFS and scaling the number of active cores (DVFCS) has been proposed. Compared to DVFS, DVFCS is capable of achieving higher energy efficiency by balancing dynamic and leakage power savings. Jian *et al.* addressed the problem of finding a chip-wide operating voltage and frequency setting as well as the number of active cores that minimizes the power dissipation of a general-purpose chip multiprocessor under performance constraints [82]. Lee *et al.* studied to maximize the performance of power constrained GPUs by coordinating the number of active cores and chip-wide voltages/frequencies of both cores and caches [83]. However, all the research stated above assume that the DVFS is on the chip level.

This chapter addresses the problem of minimizing the power dissipation of many-core systems under performance constraints by exploiting per-core DVFS with core scaling, and shows that per-core DVFCS can achieve significant higher energy-efficiency compared with traditional DVFS methods [84]. The major contribution of the work includes: 1) detail analysis and description of the proposed algorithm; 2) more analysis on both dynamic power and leakage power saving from DVFCS, respectively, for different performance constraints; 3) the sensitivity of energy efficiency improvement and optimal core count to different model parameters, including the number of global voltage supplies and leakage power ratio (different CMOS process). The reminder of the chapter is organized as follows: Section 5.2 states the problem in a formal mathematical way. Section 5.3 describes the proposed algorithm. Section 5.4 presents the models and benchmarks used in our experiments. Section 5.5 discusses the experimental results with detailed analysis. Finally, Section 5.6 concludes the chapter.

5.2 Problem Formulation

An application is required to run at a throughput of T_{REQ} on a many-core system which has $CORE_{AVL}$ cores available, and N global voltage supplies (V_1, V_2, ..., V_N) in ascending order, where V_i is chosen from a set of available discrete voltage levels. There are a set of implementations (A_1, A_2, ..., A_M) for the application. Each implementation requires $CORE_{A_i}$ cores. Each mapping result may include multiple instances for each implementation, and can be represented as

$$Map = N_1 A_1 + N_2 A_2 ... + N_M A_M \tag{5.1}$$

where N_i is the number of instance of implementation A_i in the mapping result. The throughput of each implementation is determined by the operating frequency F_{A_i} of its performance critical cores, and different instance of one implementation could run at different F_{A_i}. The throughput of the mapping result T_{MAP} is given by

$$T_{Map} = \sum_{n=1}^{N_1} T_{A_1 n} + \sum_{n=1}^{N_2} T_{A_2 n} ... + \sum_{n=1}^{N_M} T_{A_M n} \tag{5.2}$$

$T_{A_i n}$ is the throughput of A_i's nth instance, and can be obtained as

$$T_{A_i n} = TC_{A_i} \times F_{A_i n} \tag{5.3}$$

where TC_{A_i} is the number of bits that can be processed by A_i per cycle. Similarly, the total power of the mapping result P_{Map} can be calculated as follows:

$$P_{Map} = \sum_{n=1}^{N_1} P_{A_1 n} + \sum_{n=1}^{N_2} P_{A_2 n} ... + \sum_{n=1}^{N_M} P_{A_M n} \tag{5.4}$$

where $P_{A_i n}$ is the power of the nth instance of A_i. For each A_i's instance, the power P_{A_i} is derived as

$$P_{A_i} = \sum_{j=1}^{CORE_{A_i}} P_{COREj}(F_{COREj}, V_{COREj}) \tag{5.5}$$

where P_{COREj} is the power consumption of each individual core in A_i, as a function of the frequency F_{COREj} and voltage V_{COREj}. The performance critical cores are required to run at $F_{COREj} = F_{A_i}$ to guarantee the throughput shown in Eq. 5.3. Noncritical cores can scale down their running frequency depends on their own workload. V_{COREj} is the minimum voltage level V_i from the global discrete voltages (V_1, V_2, ..., V_N), which satisfies $Freq(V_i) >= F_{COREj}$. The P_{COREj} is approximated as follows:

$$\begin{aligned} P_{COREj}(F_{COREj}, V_{COREj}) &= P_{DYN} + P_{LEAK} \\ &= C_{EFF} \cdot F_{COREj} \cdot V_{COREj}^2 + I_{LEAK}(V_{COREj}) \cdot V_{COREj} \end{aligned} \tag{5.6}$$

where C_{EFF} is the effective switching capacitance for each core, and $I_{LEAK}(V_{COREj})$ is the leakage current that depends on V_{COREj}.

In sum, the problem can be formulated as:

$$\begin{aligned} &minimize && P_{Map} \\ &\text{subject to} && T_{REQ} \leq T_{Map} \\ & && CORE_{AVL} \geq \sum CORE_{A_i} \cdot N_i \end{aligned}$$

by exploring (1) the number of instance of each implementation (N_1, N_2, ..., N_M), (2) the throughput of each implementation instance $T_{A_i n}$, and (3) the global voltage levels (V_1, V_2, ..., V_N).

Fig. 5.1 illustrates an example of the problem. There are three implementations (A_1, A_2, A_3) in the application library. A many-core platform with N global voltage supplies is given. There could be multiple mapping results that satisfy the performance requirement while fitting on the given many-core system. The goal is to select the most energy efficient mapping result, i.e. ($2A_2 + A_3$) in this example, while setting the global voltage levels as (V_1, V_2, ..., V_N) to minimize the power dissipation.

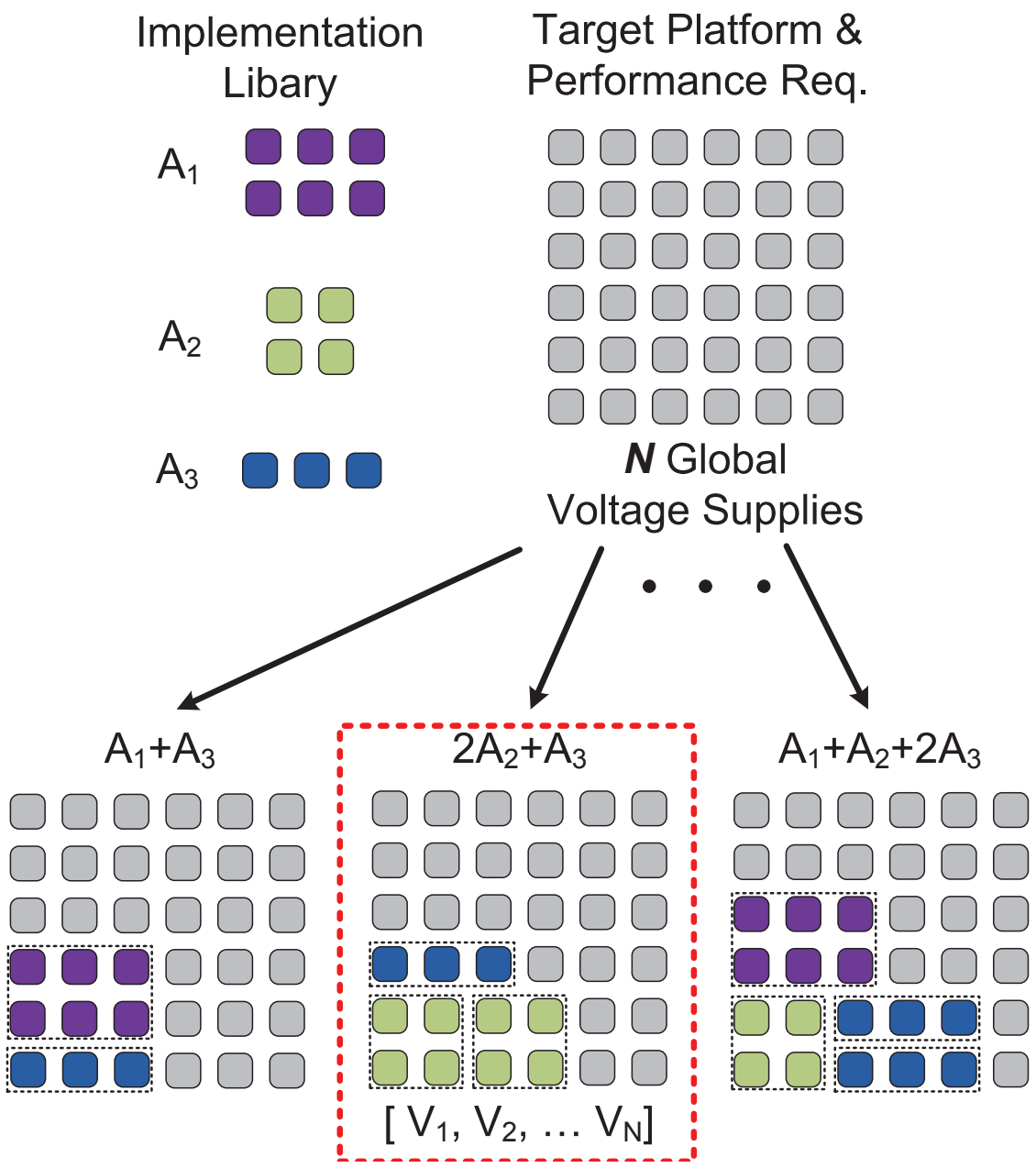
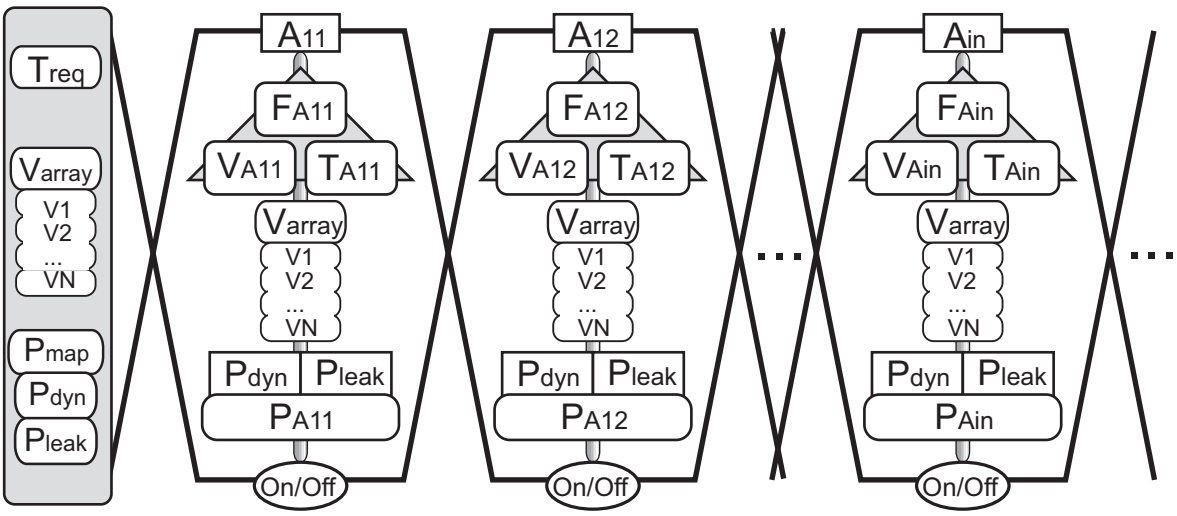


Figure 5.1: There are three different implementations (A_1, A_2, A_3) for the application to be mapped. With a given many-core platform and a performance requirement, multiple mapping results are generated initially. Finally, the most energy efficient mapping result ($2A_2 + A_3$) is chosen, and the N global voltage supplies are set as (V_1, V_2, ..., V_N) to minimize the power dissipation. V_i is chosen from a set of available discrete voltage levels.



(a) Block diagram of the chromosome in GA

(b) Solution example

Figure 5.2: Chromosome block diagram and solution example of the proposed GA. Only shaded implementation instances are included in the solution.

5.3 Proposed Algorithm

Due to the large search and optimization space, a heuristic based on genetic algorithms (GA) is proposed to solve the problem. GA is based on a mechanism of natural selection and evolutionary genetics. Each solution of the problem is represented as a *chromosome*, which contains a sequence of *genes*. A *fitness* value is associated with each solution. The fitness value shows how close the solution is to the optimum. The process starts with an initial *population* of solutions created in some way, *e.g.,* randomly. In each iteration, new offspring solutions are generated through *selection*, *crossover* and *mutation*. Then, some new solutions are added to, and some old solutions are removed from the *population* based on their *fitness*. The evolution process stops when either the optimization goal or the maximum number of iterations is reached [137].

The block diagram of a chromosome in the proposed GA is shown in Fig. 5.2(a). Each gene represents an implementation instance A_in with a particular running frequency F_{A_in}. There is one "On/Off" bit for each gene to indicate whether it is included in the final solution. The number

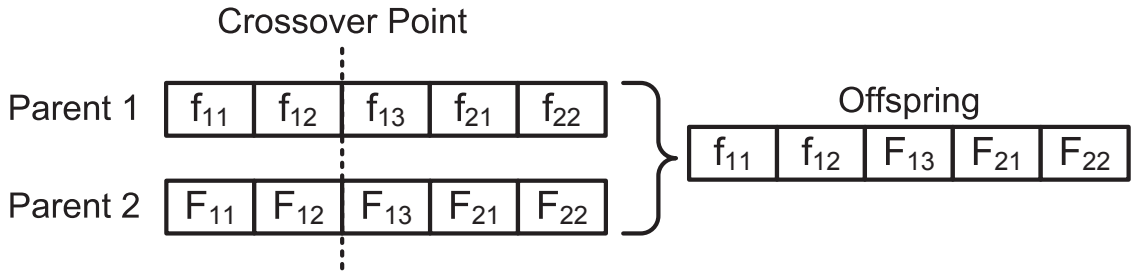
of genes for each implementation A_i in one chromosome is determined by the maximum number of A_i can fit on the given many-core platform, *i.e.,* $CORE_{VAL}/CORE_{A_i}$. For example, there are two implementations (A_1, A_2) for the targeted application, which occupies two and three cores, respectively. And there are six cores available on the targeted many-core system. Therefore, there are three instances of A_1 and two instances of A_2 in each chromosome, as shown in Fig. 5.2(b). Additionally, only the implementation instances being set to "On" are included in the final solution. In the above example, A_{11} and A_{21} are selected in the final solution. Therefore, the mapping result uses five cores in total.

Algorithm 8 . *Pseudocode of the Proposed Algorithm*

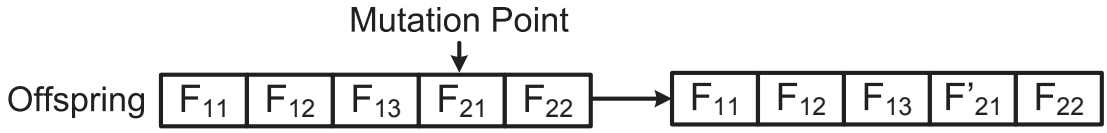
1: Ω = Create_Initial_Population();
2: **for** Each *chrom* in Ω **do**
3: Compute_Fitness(*chrom*);
4: **end for**
5: $iterNum = 0$;
6: **while** $iterNum < MaxIterNum$ **do**
7: $\{Parent_1, Parent_2\}$ = Parents_Selection(Ω);
8: $Offspring$ = Crossover($Parent_1, Parent_2$);
9: Mutation(*Offspring*);
10: Pow = Compute_Fitness(*Offspring*);
11: **if** Pow is less than the maximum power in Ω **then**
12: Replace(*Offspring*, Ω);
13: **end if**
14: $++iterNum$;
15: **end while**
16: **return** $minPower$, σ and $seqOn$ of the *chrom* which dissipates the minimum power in Ω;
17: (Continued on next page.)

The pseudocode of the proposed algorithm is presented in Algorithm 8. The initial population Ω is created randomly to increase both the diversity of the population and the chance to explore global optima. During the initial population creation, the operation frequency of per-

```
18: procedure COMPUTE_FITNESS(chrom)
19:     minPower = INT_MAX;
20:     Global Voltage Selection σ = (0, 0, ..., 0);
21:     Gene "On" bit sequence seqOn = (0, 0, ..., 0);
22:     for Different Global Voltage Settings: λ do
23:         Reset "On" bits in chrom;
24:         Calculate power for each gene in chrom;
25:         Use greedy knapsack to turn "On" genes, get sol and calculate its power pow;
26:         if pow < minPower then
27:             minPower = pow;
28:             σ = λ;
29:             seqOn = sol;
30:         end if
31:     end for
32:     return minPower, σ and seqOn along with chrom;
33: end procedure
```



(a) Crossover



(b) Mutation

Figure 5.3: Procedure and example of (a) crossover and (b) mutation.

formance critical cores $F_{A_i n}$ in each chromosome is assigned randomly. Since the object of the algorithm is to minimize power dissipation, the fitness value of a solution is inversely proportional to the power dissipation, and defined as

$$fitness_{chrom} = 1/Power_{chrom} \tag{5.7}$$

After the population is initialized, the fitness value is computed for each chromosome in the population. For a given global voltage setting, the power and throughput of each gene can be calculated based on its F_{A_i}. Then, the chromosome is sent to a greedy knapsack solver to determine which implementation instance should be included (turned "On"), and which not (turned "Off") in the final solution. After iterating all possible global voltage settings, the solution with the lowest power dissipation is returned.

In each iteration of GA, two chromosomes are selected as parents for crossover. During parents selection, a roulette wheel strategy is applied. The chance for each chromosome to be selected in each iteration is proportional to how its fitness value compared with other chromosomes, and defined as

$$Pr_{chrom} = \frac{fitness_{chrom}}{\sum_{j=1}^{N} fitness_j} \tag{5.8}$$

Chromosomes with higher fitness value have more probability of selection. The selected parent chromosomes are sent to crossover and mutation to generate offsprings. As shown in Fig. 5.3(a), a

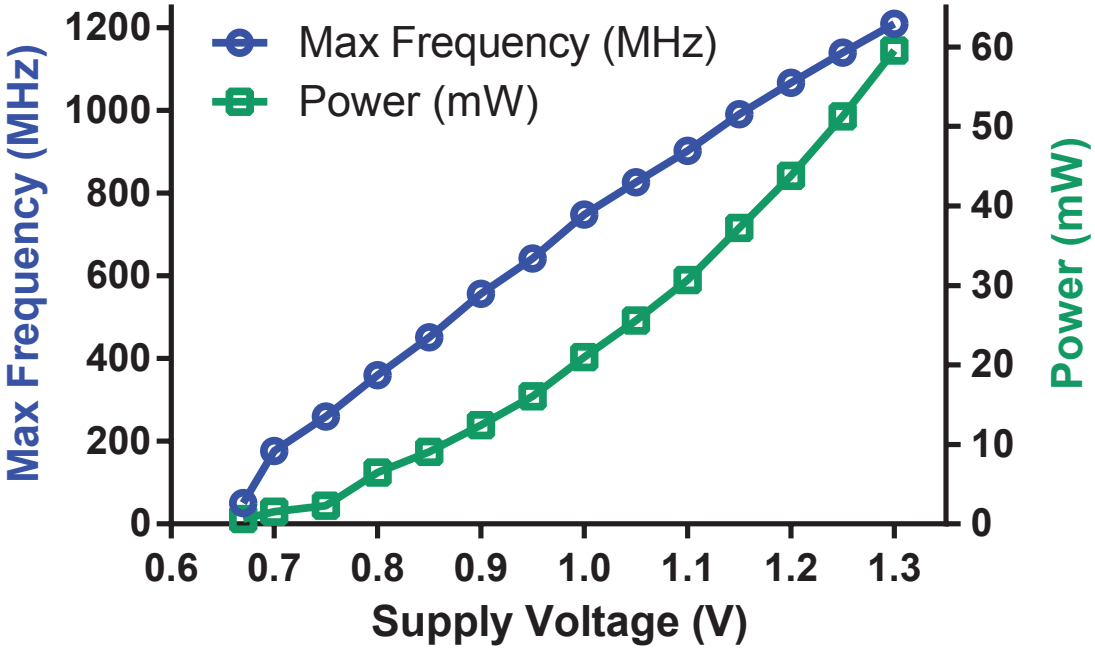
random point is chosen to break each parent into two parts during crossover. Then the offspring is generated by coping the part before the crossover point from Parent-I and part after the crossover point from Parent-II. After crossover, the generated offspring is sent into a mutation operator. The mutation operator is crucial to push the algorithm toward unexplored regions of the search space, therefore avoiding local optima. Fig. 5.3(b) shows the procedure of the mutation operation. A random gene (implementation instance) is selected and assigned with a random operation frequency. Finally, the offspring is sent to the fitness calculation process to generate a valid solution. If the generated solution consumes less power than one of the chromosomes in the population set, the chromosome with weakest fitness (the solution with largest power dissipation) in the set is replaced by the offspring for the next iteration. When the maximum number of GA iteration is reached, the algorithm returns the chromosome with the least power dissipation in the population set as the final solution.

5.4 Methodology

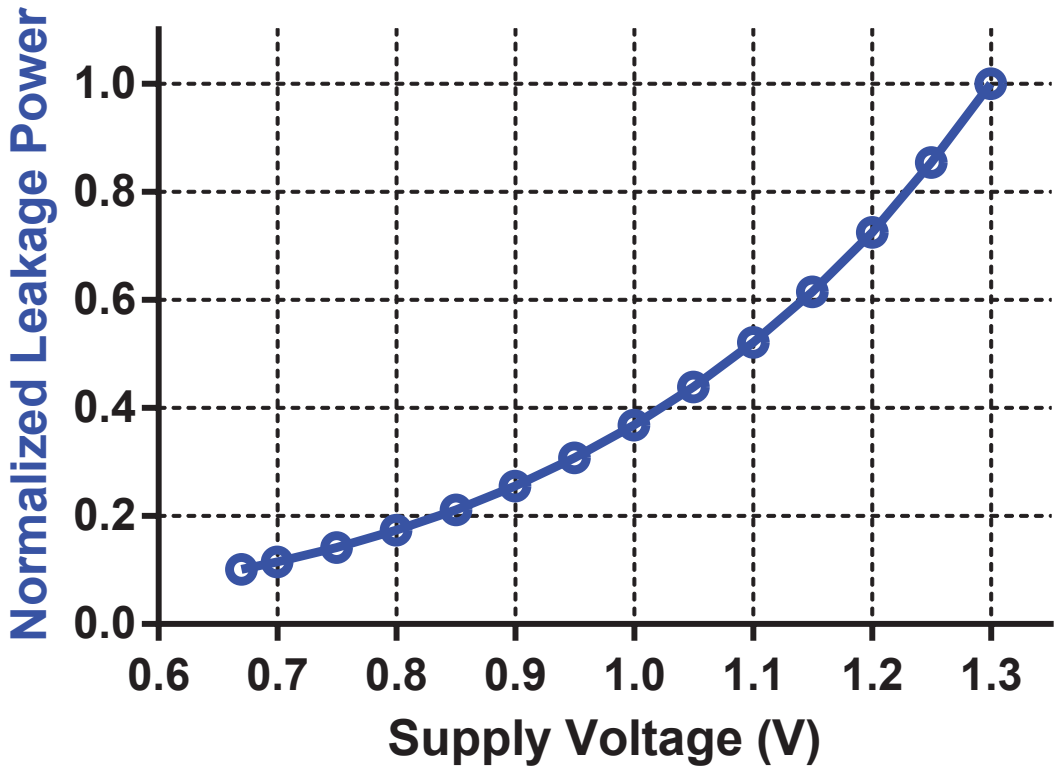
5.4.1 Power Model

Each core on AsAP can operate up to 1.2 GHz at 1.3 V [91]. The maximum operation frequency and dynamic power dissipation of cores on AsAP have a near-linear and quadratic dependence on the supply voltage, as shown in Fig. 5.4(a). The data are measured from real-silicon chip, and used for the simulations in the next section.

Since AsAP was fabricated with super low-power CMOS process, the leakage power from the chip measurement is negligible. To simulate processors with different leakage ratio $LR = P_{LEAK}/P_{DYN}$ at 1.3 V, the leakage power scaling factors for different voltage supplies are generated from a dummy circuit with ST 65 nm CMOS technology. The dummy circuit is composed of a large number of NAND, NOR and INV gates. Each gate has different number of inputs (1 to 4) and the input states are randomly selected [131]. The normalized leakage power scaling factors are measured from the HSPICE simulation and shown in Fig. 5.4(b).



(a) Maximum Frequency and Dynamic Power



(b) Normalized Leakage Power

Figure 5.4: (a) Maximum operation frequency and dynamic power of one core; (b) Normalized leakage power scaling factor.

Table 5.2: Number of cores and throughput of different AES engines

AES Implementations	Cores Usage	1/Throughput (cycles/byte)
Small	8	167.375
One-task one-processor	9	223.875
Parallel-MixColumns	15	136.250
Parallel-SubBytes-MixColumns	18	84.375
Loop-unrolled Three Times	23	68.625
Loop-unrolled Nine Times	50	16.625
No-merge-parallelism	59	9.500
Full-parallelism	137	4.375

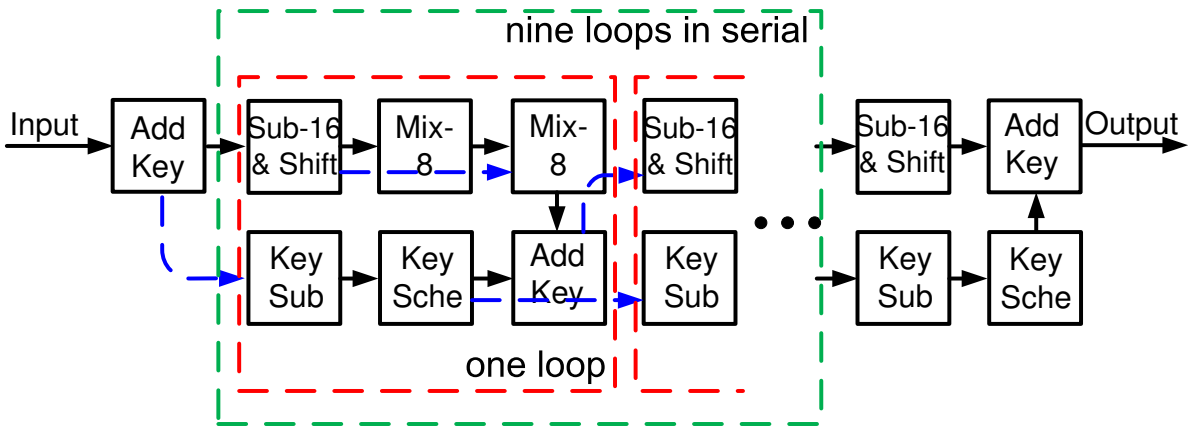


Figure 5.5: 59-core AsAP mapping of the No-merge-parallelism implementation [38].

5.4.2 Benchmark – Eight AES Engines

An implementation library with eight Advanced Encryption Standard (AES) engines is selected as the benchmark. Table 5.2 lists the number of cores and throughput for different AES implementations. The smallest implementation only occupies eight cores, while the largest one requires 137 cores. The throughput of the eight AES engines at 1.3 V and 1.2 GHz are ranged from 58 Mbps to 2.2 Gbps [40].

5.5 Experimental Results

The experiment results are discussed in this section. First, we compare the proposed per-core DVFCS mechanism with traditional per-core DVFS in terms of power saving effectiveness. Later, the impact of changing the number of global voltage supplies (V_{DD}) and the leakage ratio (LR) of the processor on the energy efficiency improvement brought by core scaling and the optimal core count are also investigated.

5.5.1 Per-Core DVFCS VS. Per-Core DVFS

Without losing generality, we assume that there are two global V_{DD}s on the targeted many-core system, and the leakage ratio $LR = 30\%$. Three different voltage, frequency, and core scaling configurations are studied in this subsection:

- No DVFS;
- Per-core DVFS with two global V_{DD}s (TV);
- Per-core DVFCS with two global V_{DD}s (TVC).

Configurations with different number of V_{DD}s and LR are investigated later. The *NoDVFS* is selected as baseline to evaluate the power saving efficiency of other configurations. One No-merge-parallelism engine is selected if core scaling is not applicable, since it is the most energy-efficient implementation compared with others [40]. The maximum throughput (Th_{MAX}) of one No-merge-parallelism engine is approximate to 1 Gbps. Fig. 5.5 shows the mapping graph of the 59-core No-merge-parallelism AES engine.

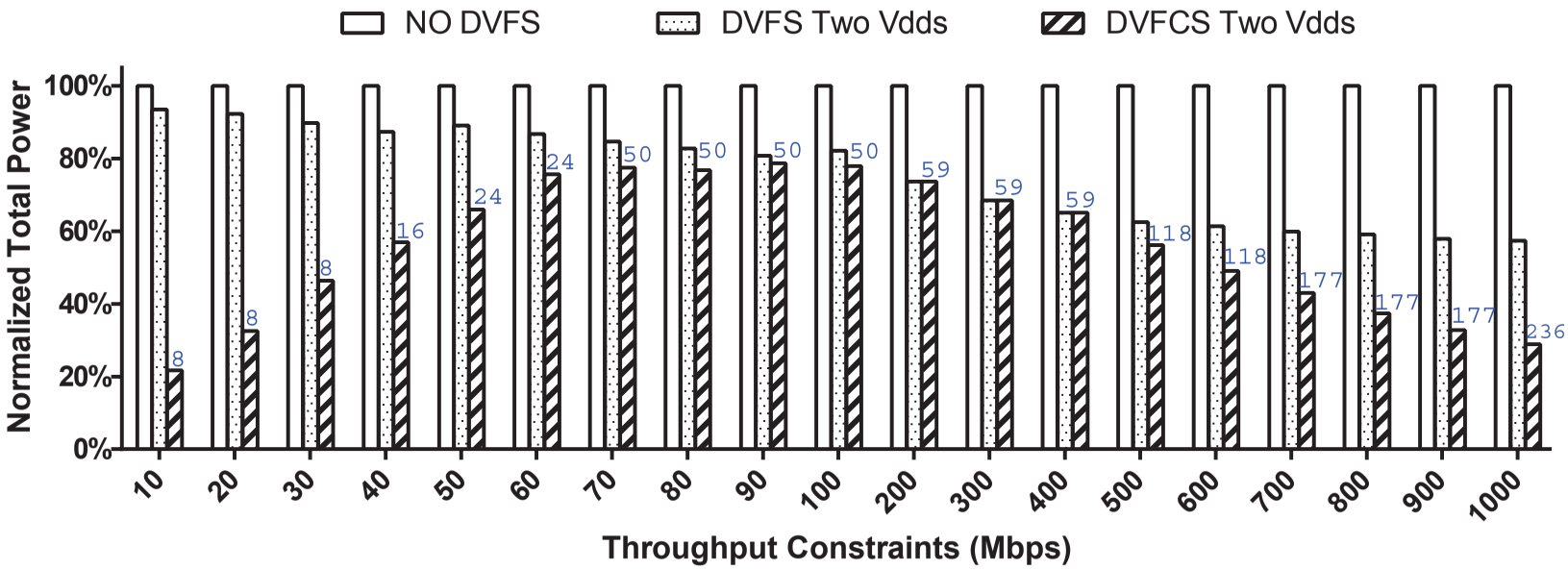


Figure 5.6: Normalized power dissipation of different voltage/frequency/core scaling configurations versus throughput constraints. The leakage power ratio $LR = 30\%$. *NoDVFS*, and *DVFS Two Vdds* use single *No-merge-parallelism* engine. The optimal core count chosen by *DVFCS Two Vdds* are shown on the top of power bars.

Fig. 5.6 shows the power dissipation for different configurations, all normalized to *NoD-VFS*. With two V_{DD}s, DVFCS outperforms DVFS for all different throughput requirements. TVC can save as much as 72% extra power compared with TV. The extra power saving brought by core scaling clearly shows a non-linear relationship with the throughput constraints. To better understand the fundamental reasons of the non-linear relationship, we divide the simulation results into three categories, which are systems with (1) a loose ($\leq 20\%Th_{MAX}$), (2) an intermediate ($20\% \sim 80\%Th_{MAX}$), and (3) a tight ($\geq 80\%Th_{MAX}$) performance constraint.

As shown in Fig. 5.7, ∼75% power dissipation is due to leakage (P_{leak}) when the system has a loose performance constraint. As the performance constraint increases into the intermediate and tight range, dynamic power (P_{dyn}) starts to dominate in the total power dissipation. Fig. 5.8 shows the dynamic and leakage power saving from DVFS and DVFCS for different throughput constraints, respectively. As discussed in Section 5.1, traditional DVFS is aimed for reducing P_{dyn} by scaling down voltage and frequency. Although scaling down voltage can also reduce P_{leak}, it is not as effective as dynamic power reduction. As shown in Fig. 5.8, DVFS always saves more P_{dyn} than P_{leak} no matter the throughput requirement is loose or tight. As a result, even P_{leak} is significant higher than P_{dyn} in the total power consumption, DVFS still tries to save from P_{dyn}, which impairs its power saving effectiveness when the performance constraint is loose. On the other hand, core scaling always try to save more from the dominate part of the power, which makes it favorable for all the performance constraint range. When the throughput requirement is loose, core scaling obtains its power saving mostly from P_{leak}. As the throughput requirement increases, the majority of power saving starts to shift to P_{dyn}. As shown in Fig. 5.8, core scaling even shows negative dynamic power saving at a few loose throughput constraints, which proves that DVFCS tries to minimize the total power dissipation rather than only aiming for a specific part of the power.

Fig. 5.6 also shows that the optimal core count chosen by DVFCS tends to increase along with the throughput constraint. With a loose throughput constraint, core scaling selects implementations with small number of cores in order to reduce P_{leak}. As a result, TVS reduces power dissipation by 28% compared with TV, as shown in Table 5.3. As the performance constraint gets into the intermediate range, P_{dyn} starts to dominate. Therefore, the extra power saving introduced by core scaling is decreased to 7%. When the performance target is close to Th_{MAX},

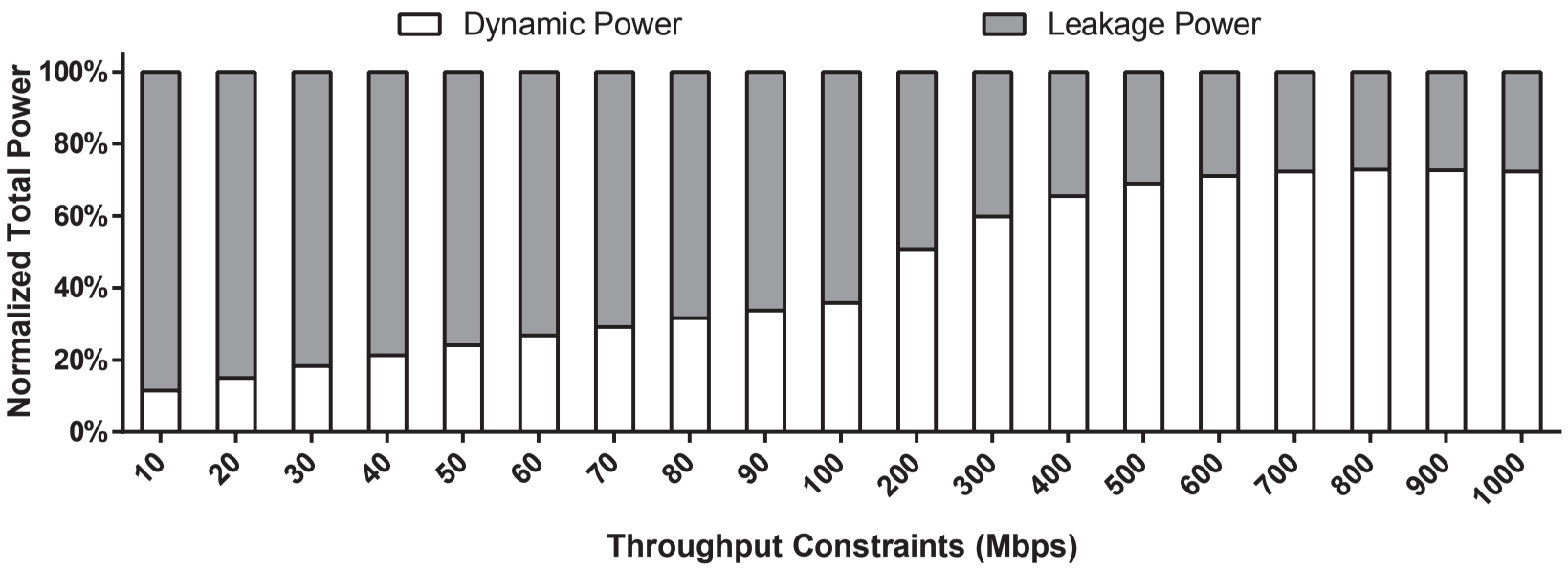


Figure 5.7: Dynamic and leakage power partition in percentage of *NoDVFS* with one *No-merge-parallelism* engine versus throughput constraints. The leakage power ratio $LR = 30\%$.

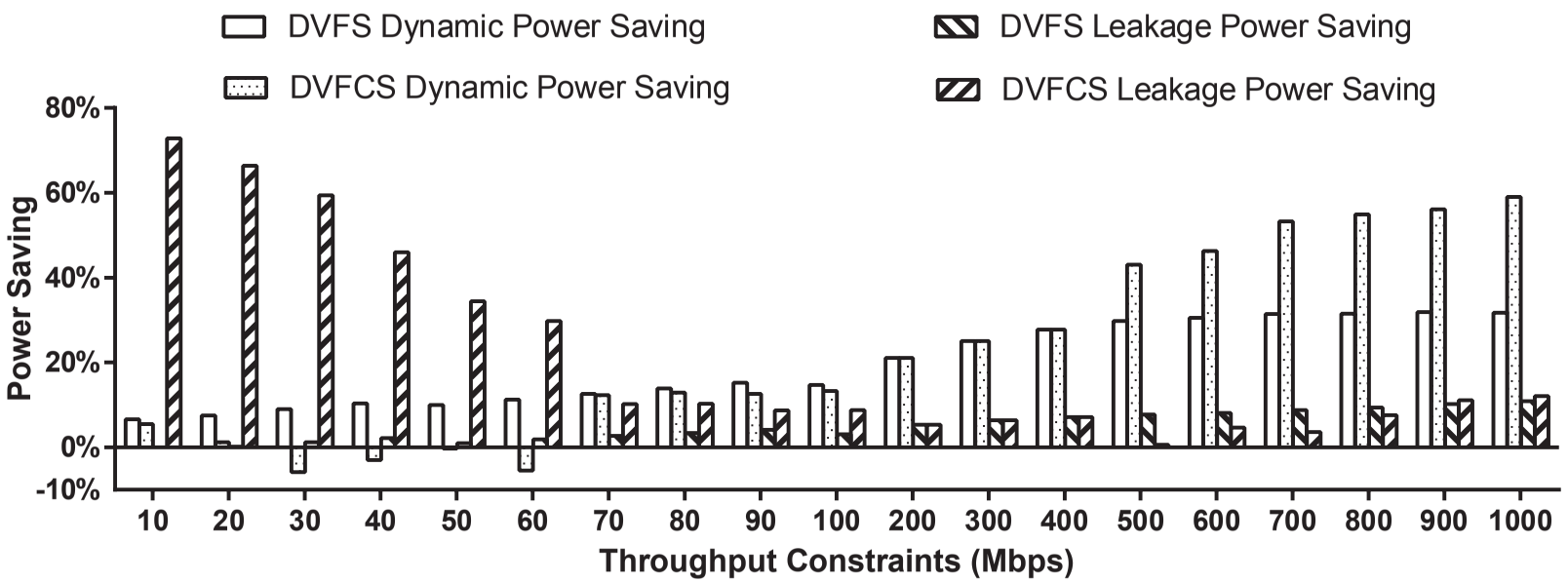


Figure 5.8: Dynamic and leakage power saving in percentage of *DVFS* and *DVFCS* with two V_{DD}s versus throughput constraints. The leakage power ratio $LR = 30\%$.

core scaling starts to apply multiple instances for each implementation. As a result, each instance requires a lower voltage/frequency level compared to TV, which causes 25% less power dissipation.

5.5.2 Different Number of Global Voltage Supplies

In this subsection, we analyze the impact of the number of V_{DD}s on the experiment results. Without modifying the leakage ratio ($LR = 30\%$), four more voltage/frequency/core scaling configurations are studied:

- Per-core DVFS with one global V_{DD}s (OV);
- Per-core DVFS with three global V_{DD}s (THV);
- Per-core DVFCS with one global V_{DD}s (OVC);
- Per-core DVFCS with three global V_{DD}s (THVC);

Fig. 5.9 shows the optimal core count selected by per-core DVFCS for systems with different number of V_{DD}s versus performance requirements. Since the optimal core counts are discrete numbers, curve fitting lines are also plotted to help illustrate the results. As the throughput constraint becomes tighter, DVFCS tends to utilize more active cores by mapping multiple instances for each implementation in order to save P_{dyn}. Additionally, DVFCS uses fewer active cores for the same throughput constraint when there are more V_{DD}s available for the targeted many-core system. It implies that without impairing energy efficiency, a mapping result could be shrunk to fit in less number of cores by adding extra V_{DD}s. THVC uses 22% and 9% fewer active cores in average than OVC and TVC, respectively.

Fig. 5.10 shows the extra power saving in percentage brought by per-core DVFCS for systems with one, two and three V_{DD}s, respectively. The non-linear relationship between the power saving improvement and throughput constraints exists consistently no matter how many V_{DD}s are available. DVFCS tends to improve more on energy efficiency when the throughput is either loose or tight. As the number of V_{DD}s increases, the power saving improvement brought by core scaling tends to increase when the throughput requirement is loose, while decrease when the throughput requirement is close to Th_{MAX}.

Table 5.3: Average power savings in percentage for DVFS and DVFCS with different number of V_{DD}s compared with *NoDVFS*. The leakage power ratio $LR = 30\%$.

Configurations		Loose Perf. Target			Inter. Perf. Target			Tight Perf. Target		
		Dyn. Saving	Leak. Saving	Total Saving	Dyn. Saving	Leak. Saving	Total Saving	Dyn. Saving	Leak. Saving	Total Saving
DVFS with One Vdd		9.2%	0.0%	9.2%	17.1%	0.0%	17.1%	17.8%	0.0%	17.8%
DVFS with Two Vdds		11.5%	2.1%	13.6%	28.9%	7.6%	36.5%	31.7%	10.2%	41.9%
DVFS with Three Vdds		12.0%	2.3%	14.4%	33.4%	8.9%	42.3%	38.2%	12.5%	50.7%
DVFCS with One Vdd		4.8%	25.4%	30.2%	35.7%	-3.9%	31.8%	53.4%	6.5%	59.9%
DVFCS with Two Vdds		5.9%	35.6%	41.5%	39.1%	4.5%	43.6%	56.7%	10.3%	67.0%
DVFCS with Three Vdds		5.7%	37.5%	43.1%	41.0%	6.1%	47.1%	57.6%	11.4%	69.0%
Extra Power	One Vdd	-4.4%	25.5%	21.1%	18.6%	-3.9%	14.7%	35.6%	6.5%	42.1%
Saving From	Two Vdds	-5.6%	33.5%	27.9%	10.2%	-3.1%	7.1%	25.0%	0.1%	25.1%
Core Scaling with	Three Vdds	-6.3%	35.2%	28.9%	7.6%	-2.8%	4.8%	19.4%	-1.1%	18.3%

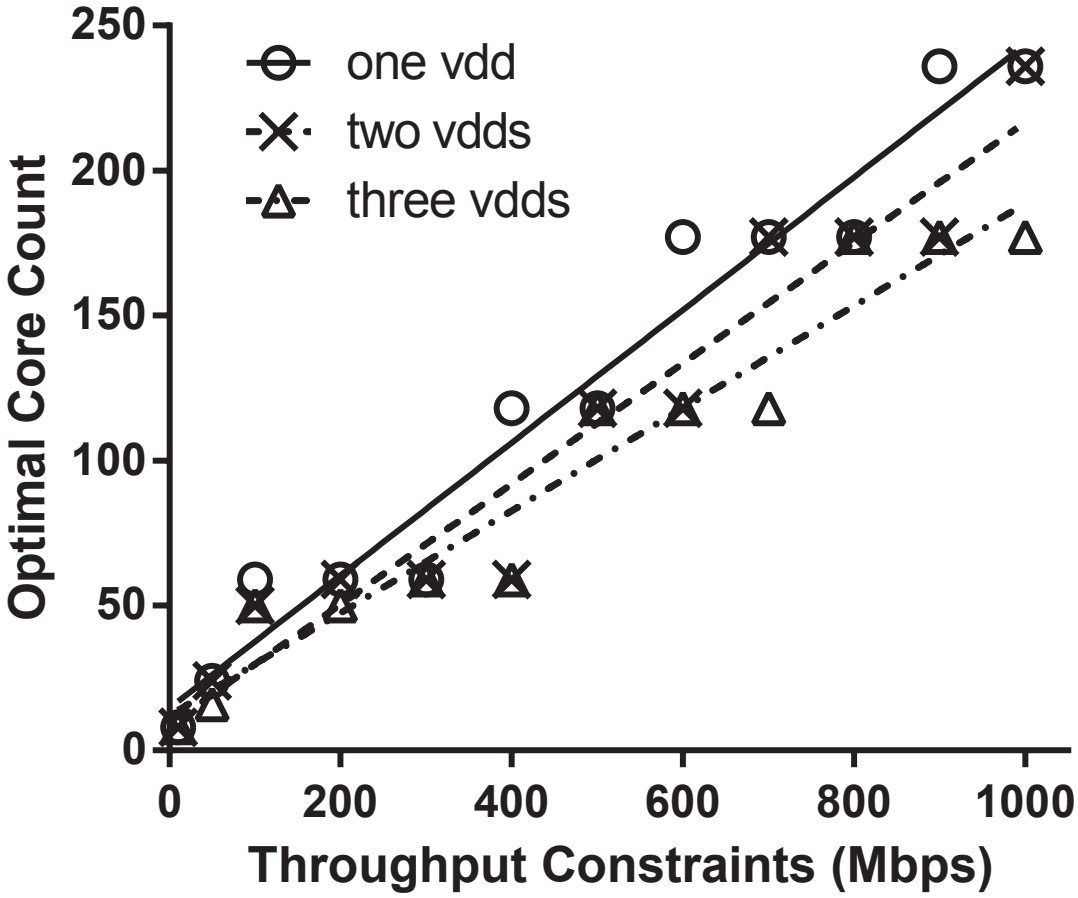


Figure 5.9: Optimal core count selected by DVFCS with different number of V_{DD}s versus throughput constraints.

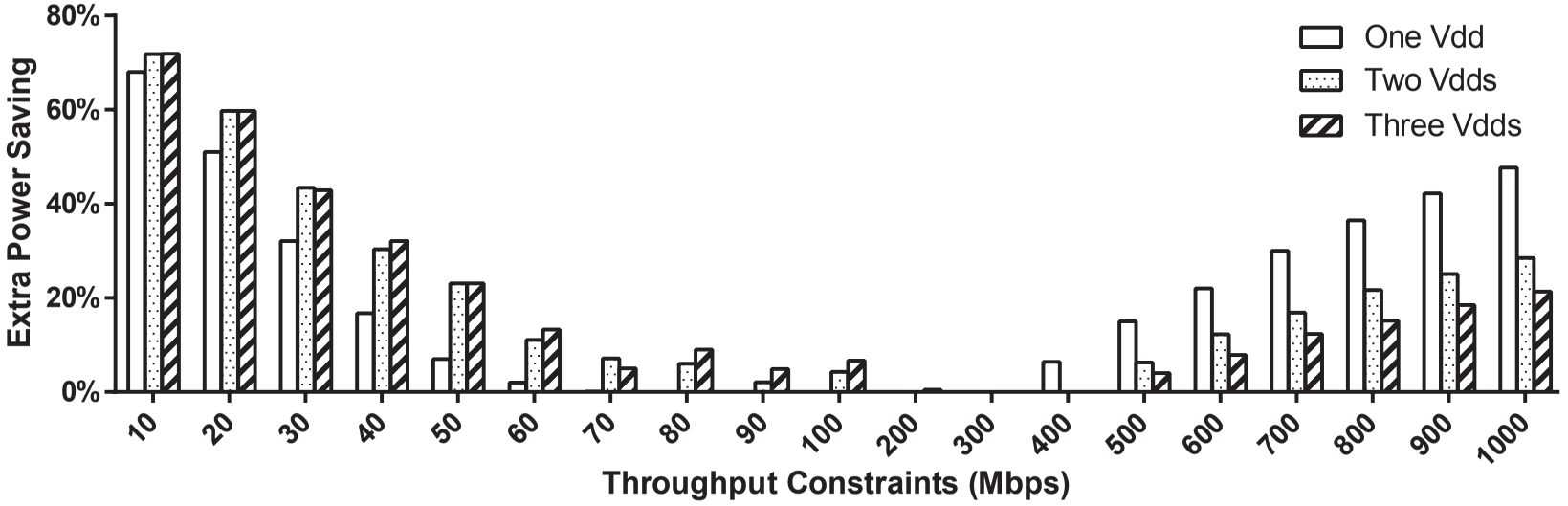


Figure 5.10: Extra power saving in percentage introduced by DVFCS compared with DVFS with one, two and three V_{DD}s versus throughput constraints.

Table 5.3 lists the average power saving of six voltage/frequency/core scaling configurations for three different categories of performance constraints. With a loose performance requirement, per-core DVFCS brings 21%, 28% and 29% extra power saving compared with traditional DVFS for systems with one, two and three V_{DD}s, respectively. The extra power saving is larger when there are more V_{DD}s. In such a case, P_{dyn} occupies a relative small portion in total power dissipation. As a result, the benefit brought by additional V_{DD}s is limited for traditional DVFS. On the other hand, a large number of V_{DD}s offers per-core DVFCS more flexibility to optimize the total power dissipation. For example, the P_{dyn} deficit increases from 4.4% to 6.3% when the number of V_{DD}s changes from one to three. However, THVC saves extra 10% power from P_{leak} compared with OVC, which overcomes the deficit from P_{dyn} and leads to 8% improvement in total power dissipation.

With a tight performance constraint, per-core DVFCS shows 42%, 25% and 18% energy efficiency improvement compared with DVFS for systems with one, two and three V_{DD}s, respectively. The power saving improvement declines as the number of V_{DD}s increases, which is a complete opposite compared with the situation when the performance constraint is loose. As we discussed earlier, per-core DVFCS tends to use fewer active cores when there are more V_{DD}s. It implies that the active cores in THVC are required to run at higher frequency than OVC and TVC to satisfy the same performance requirement, which diminishes the extra P_{dyn} saving brought by adding extra V_{DD}s. Therefore, DVFS is more beneficial in terms of improving energy efficiency by increasing the number of V_{DD}s. As shown in Table 5.3, THV improves the P_{dyn} by 25% and 6% compared with OV and TV, while THVC only saves 4% and 1% more on P_{dyn} compared with OVC and TVC, respectively.

5.5.3 Different Leakage Ratio

In order to investigate the impact of leakage ratio on optimal solutions, four different leakage ratios are studied in this subsection, including 30%, 20%, 10% and 5%. The leakage ratio range covers from high-performance process that is very leaky, to the process aiming for low power. In this subsection, we assume that there are two global voltage supplies for the targeted many-core system.

Fig. 5.11 shows the optimal core count selected by per-core DVFCS for different leakage

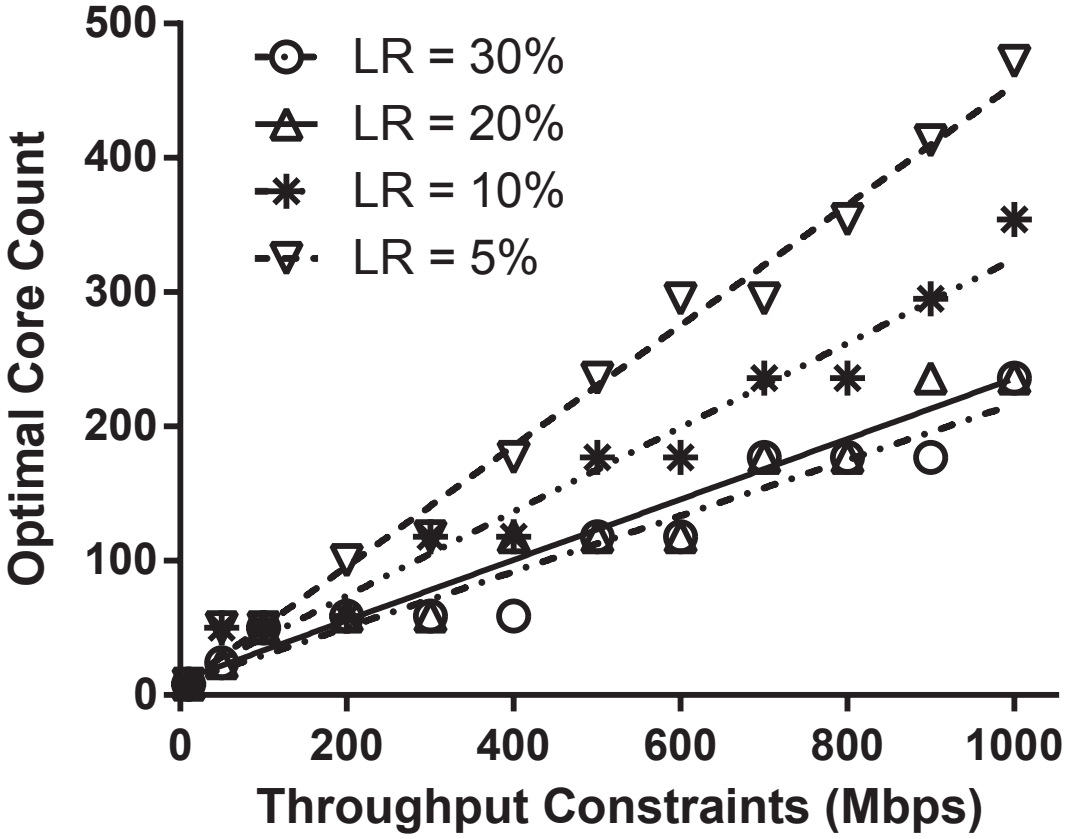


Figure 5.11: Optimal core count selected by DVFCS versus throughput constraints for different leakage ratios.

ratio. As expected, the leakage power becomes more dominant along with the increment of leakage ratio, thus favoring usage of fewer active cores. As a result, LR30 utilizes 10%, 31% and 50% fewer active cores than LR20, LR10 and LR5, respectively.

The power saving improvement in percentage brought by per-core DVFCS for systems with different leakage ratio is shown in Fig. 5.12. DVFCS tends to save more extra power than DVFS when the throughput is approaching loose or tight for all four different leakage ratio. The non-linear relationship between the power saving improvement and throughput constraints exists across from high-performance to low-power CMOS technology. When the performance requirement is loose, DVFCS introduces more extra power saving when the leakage ratio is high. On the other hand, when the performance requirement is close to Th_{MAX}, the extra power saving brought by DVFCS favors less leaky process.

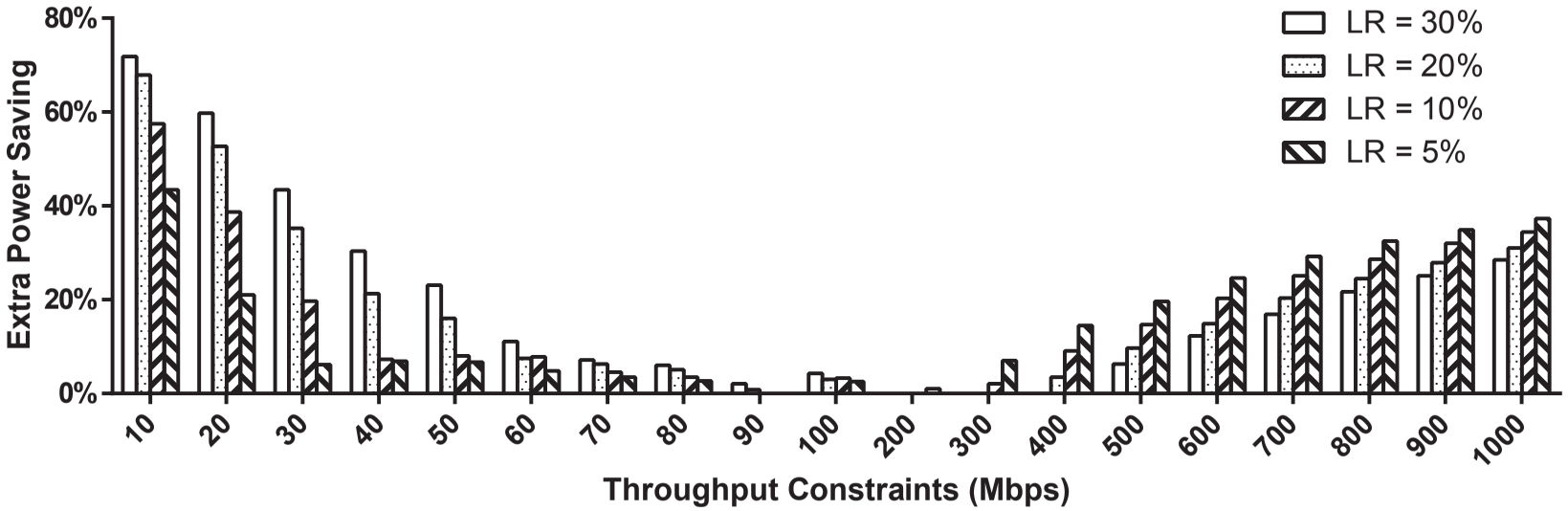


Figure 5.12: Extra power saving in percentage introduced by two-V_{DD} DVFCS compared with two-V_{DD} DVFS versus throughput constraints for different leakage ratios.

Table 5.4: Average power savings in percentage for DVFS and DVFCS with different leakage ratios compared with *NoDVFS*. Two V_{DD}s are available for the targeted many-core system.

Configurations		Loose Perf. Target			Inter. Perf. Target			Tight Perf. Target		
		Dyn. Saving	Leak. Saving	Total Saving	Dyn. Saving	Leak. Saving	Total Saving	Dyn. Saving	Leak. Saving	Total Saving
DVFS with 30% LR		11.5%	2.1%	13.6%	28.9%	7.6%	36.5%	31.7%	10.2%	41.9%
DVFS with 20% LR		15.1%	1.8%	16.9%	32.4%	5.7%	38.1%	34.9%	7.5 %	42.4%
DVFS with 10% LR		22.2%	1.3%	23.5%	36.8%	3.2%	40.0%	38.8%	4.2 %	43.0%
DVFS with 5% LR		29.6%	0.8%	30.4%	39.6%	1.7%	41.3%	41.1%	2.2 %	43.3%
DVFCS with 30% LR (LR30)		5.9 %	35.6%	41.5%	39.1%	4.5%	43.6%	56.7%	10.3%	67.0%
DVFCS with 20% LR (LR20)		10.3%	30.6%	41.0%	46.2%	1.6%	47.8%	63.2%	6.9%	70.1%
DVFCS with 10% LR (LR10)		18.6%	22.8%	41.4%	56.9%	-2.5%	54.3%	72.3%	2.2%	74.6%
DVFCS with 5% LR (LR5)		30.0%	13.1%	43.1%	63.5%	-3.2%	60.3%	78.3%	-0.1%	78.2%
Extra Power Saving From Core Scaling with	30% LR	-5.6%	33.5%	27.9%	10.2%	-3.1%	7.1%	25.0%	0.1%	25.1%
	20% LR	-4.8%	28.8%	24.0%	13.8%	-4.1%	9.7%	28.3%	-0.6%	27.7%
	10% LR	-3.6%	21.5%	17.9%	20.1%	-5.7%	14.4%	33.5%	-2.0%	31.5%
	5% LR	0.4%	12.3%	12.7%	23.9%	-4.9%	19.0%	37.2%	-2.3%	34.9%

Table 5.4 lists the average power saving in percentage of DVFCS and DVFS with different leakage ratios for three different categories of performance constraints. With a loose performance requirement, per-core DVFCS brings 28% extra power saving compared with traditional DVFS for a high-performance leaky process ($LR = 30\%$), and 13% for a low-power less-leaky process ($LR = 5\%$). The extra power saving is larger when the leakage ratio is higher. As discussed in Subsection 5.5.1, ∼75% power dissipation is due to leakage when the performance requirement is loose. A higher leakage ratio results in the optimal solution shifts to use fewer active cores, which reduces the leakage power dissipation significantly. Although this causes increased dynamic power dissipation due to a higher operating frequency, it obtains an overall net extra improvement in energy efficiency. For example, DVFCS-LR30 consumes 6% more dynamic power compared with DVFS, and saves 34% in leakage. On the other hand, DVFCS-LR5 dissipates about the same about of dynamic power compared with its DVFS counterpart, but it only saves 12% in leakage. As a result, the energy efficiency improvement brought by DVFCS-LR5 is 15% less than DVFCS-LR30.

With a tight performance constraint, power saving improvement brought by per-core DVFCS increases from 25% to 35% when the leakage ratio declines from 30% to 5%. The extra energy efficiency improvement decreases as LR increases. When the throughput requirement approaches Th_{MAX}, the extra power improvement introduced by DVFCS is mainly from dynamic power. As shown in Fig. 5.7, P_{dyn} occupies ∼75% of the total power dissipation when $LR = 30\%$. In such a case, a lower LR makes P_{dyn} become more dominant, and leaves the core scaling algorithm more room to save in P_{dyn} by manipulating the number of active cores used in optimal solutions.

5.6 Conclusion

We have formulated and addressed the problem of minimizing the power dissipation of performance-constrained many-core processors by exploiting per-core DVFS and core scaling simultaneously. A GA-based algorithm is proposed to solve the problem. The experimental results show that optimizing the number of active core and voltage/frequency levels appropriately can improve the energy efficiency by 5% to 42% compared with per-core DVFS for a range of throughput requirements. The energy efficiency improvement shows a non-linear relationship with the performance constraint, which tends to increase when the performance constraint inclines to be either

loose or tight.

Additionally, when the throughput requirement is loose, the energy efficiency improvement brought by core scaling favors systems with more global voltage supplies (21% for OVC, and 29% for THVC) and high-performance leaky process (13% for LR5, and 28% for LR30). On the other hand, when the throughput requirement is tight, the energy efficiency favors systems with fewer global voltage supplies (42% for OVC, and 18% for THVC) and low-power low-leaky process (35% for LR5, and 25% for LR30).

Furthermore, increasing the number of global voltage supplies or the leakage ratio can reduce the optimal core count. THVC uses 22% fewer active cores than OVC, and LR30 uses 50% fewer active cores than LR5.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This dissertation explores, designs and implements new algorithms and technologies to improve the energy efficiency of fine-grained many-core processors.

First, the dissertation demonstrates that the fine-grained many-core architecture can achieve higher area and energy efficiency inherently compared with other architecture designs for applications have high task-level and data-level parallelism by proposing 16 AES engines on AsAP2. The design space and the trade-off between performance and the number of cores are examined comprehensively. The smallest design utilizes only 6 cores for offline key expansion and 8 cores for online key expansion, while the largest requires 107 cores and 137 cores, respectively. In comparison with published AES cipher implementations on general purpose processors, the designs have 3.5–15.6 times higher throughput per unit of chip area and 8.2–18.1 times higher energy efficiency. Moreover, the design shows 2.0 times higher throughput than the TI DSP C6201, and 3.3 times higher throughput per unit of chip area and 2.9 times higher energy efficiency than the GeForce 8800 GTX.

To further improve the energy efficiency of many-core processors, the dissertation proposes an online scalable hardware-based joint local and global DVFS solution driven by workload variations for many-core processors. The local algorithms is used to select the voltage and frequency pair for each individual core based on its FIFO occupancy and stall information, while the global algorithm tunes the global voltage supplies based on the workload of all active cores.

To demonstrate the effectiveness of the proposed solution, a suite of benchmarks are tested on a many-core globally asynchronous locally synchronous (GALS) platform. The experiment results show that the proposed approach can achieve near-optimal power saving under performance constraints. Different algorithms are compared in terms of power saving, voltage switching frequency and the response delay to workload variation. The impact of the number of voltage supplies and the tuning resolution of voltage regulators on the global optimization are also investigated.

Finally, the dissertation proposes to add an extra dimension on top of the DVFS algorithm, which is core scaling. The extra orthogonal dimension brings more flexibility to the algorithm, and is able to help the algorithm find more energy efficiency solutions than traditional DVFS. The dissertation addresses the problem of minimizing the power dissipation of many-core systems under performance constraints by choosing an appropriate number of active cores and per-core voltage/frequency levels. A genetic algorithm based solution is proposed to solve the problem. Experiments with real applications show that (1) dynamically scaling the number of active cores can improve the energy efficiency by 5% to 42% compared with per-core DVFS for different performance requirements; (2) core scaling favors systems with more global voltage supplies and high-performance leaky process when the performance requirement is loose, while it favors systems with fewer global voltage supplies and low-power less-leaky process when the performance requirement is tight; (3) increasing the number of global voltage supplies or leakage ratio can reduce the optimal core count by 22% and 50%, respectively.

6.2 Future Work

There are quite a few interesting research topics on improving energy efficiency on many-core processors which are worthwhile for further investigation.

- *Implementation of the Proposed DVFS.* In Chapter 4, a joint local and global DVFS algorithm is proposed and demonstrated to be effective on power saving. It would be interesting to implement and test the proposed algorithm on real silicon. There are many challenges and critical design questions to be answered for the physical implementation. For example, (1) what is the optimal implementation and size for power gates; (2) what is the optimal granularity for power gates, per-cluster, per-core or even per-functional block; (3) what is the trade

off between the number of global voltage supplies and the efficiency of power distribution networks; (4) what are the possible communication methods between individual cores and global controller, and what is the best frequency for such whole chip involved communication. To answer the above questions, more physical design experiments and quantitative analysis are required.

- *Algorithm Design for DVFCS.* The algorithm used in Chapter 5 is based on genetic algorithm. The process for fitness computation naively loops over all possible global voltage settings for each individual solutions. As the number of global voltage supplies or available voltage levels increase, the computation time increases exponentially. As a result, the proposed algorithm in this dissertation is more applicable to static analysis, while hardly to meet real-time requirements when the number of global voltage supplies and/or available voltage levels are high. A further investigation on the convex property of energy-frequency relationship, and approximate optimal solutions may lead to an algorithm with less time complexity.

- *Extension of DVFCS.* There are two major directions to extend the current DVFCS work further. The first one is to consider situations with multiple applications. For example, if the total number of cores required by the optimal mapping for each individual applications are exceed the number of available cores on the platform, what is the best strategy to balance the core usage for different applications? If the global voltage settings selected by individual applications are different, which setting should be used? Another direction is to implement variation-aware DVFCS. The PVT variation of the available cores on the platform affects the optimal mapping results. Intuitively, the optimal solution should pick a mapping with more cores if the available cores are less leaky. However, the quantitative results depend on the workload characteristic and performance constraints of the targeted application, and the variation distribution of the available cores. Additionally, an automatic variation-aware mapping tool is required for the above analysis and also an interesting research topic.

Glossary

AES *Advanced Encryption Standard.* A symmetric encryption standard selected by the U.S. National Institute of Standards and Technology based on Rijndael algorithm.

AsAP *Asynchronous Array of simple Processors.* A parallel DSP processor consisting of a 2-dimensional mesh array of very simple CPUs clocked independently with each other.

AsAP2 The second generation of AsAP chips which also includes a few specific accelerators (FFT, Viterbi, Motion Estimation) and shared memory modules. It has a reconfigurable source synchronous network supporting long-distance interconnects for processors. Per-core DVFS is also supported for dynamic power savings.

CMOS *Complementary Metal-Oxide Semiconductor.* Technology manufactured and used in most modern digital chips.

CMP *Chip Multi-processor.* A computer architecture which integrates multiple processors into a single chip to improve processor performance.

CPI *Cycles per Instruction.* Normally the CPI for pipelined processor is larger than 1 due to the pipeline hazard or missed Cache fetch.

CPU *Central Processing Unit.* A digital integrated circuit which is designed to perform general purpose computations.

DSP *Digital Signal Processing or the processors for DSP.*

DVFS *Dynamic Voltage and Frequency Scaling.* A technique allowing processors and functional blocks to dynamically change their operating voltage and clock frequency corresponding to

their workloads. Therefore, it reduces the overall power consumption without introducing any performance penalty.

DVFCS *Dynamic Voltage, Frequency and Core Scaling.* A technique allowing many-core processors or MPSoC to dynamically change their active processing elements besides operating voltage and clock frequency in order to save power consumption.

FFT *Fast Fourier Transform.* An efficient algorithm to compute the discrete Fourier transform and its inverse.

FIFO *First-In First-Out.* A buffer queue with in-order operations: the word which is written in to the buffer first will be read out of the queue first.

FO4 *Fanout 4.* A method to define the circuit delay using the delay of an inverter with 4 inverters load.

FPGA *Field-Programable Gate Array.* A digital integrated circuit which contains an array of gates which can be programmed to perform a certain function as desired.

GA *Genetic Algorithm.* A method for solving both constrained and unconstrained optimization problems based on a natural selection process that mimics biological evolution.

GALS *Globally Asynchronous Locally Synchronous.* A design methodology in which major design blocks are synchronous, but interface to other blocks asynchronously.

GPU *Graphical Processing Unit.* A specialized digital hardware unit that produces a graphical output for a computing system. It is also used for large-scale parallel computing.

H.264 A standard for video compression. It is also known as MPEG-4 part 10.

IPC *Instructions per Cycle.* IPC is the reverse of CPI.

Mbps *Megabit per Second.* A unit of data transfer rate.

MIMD *Multiple Instruction Multiple Data.* A parallel computer architecture where different instructions with different data can be executed simultaneously.

MPSoC *Multi-Processor Systems-on-Chip.* A system-on-chip which uses multiple processors, usually targeted for embedder applications.

NoC *Network on Chip.* An on-chip communication architecture which communicates between modules in the chip using switches/routes, as in the network.

PTM *Predictive Technology Model.* PTM provides accurate, customizable and predictive model files for future transistor and interconnect technologies.

PVT *Process, Voltage, Temperature.* PVT variations are fluctuations in integrated circuits due to process fabrication variations, variations in the supply voltage, and variations caused by temperature.

SIMD *Single Instruction Multiple Data.* A data parallelism technique where one single instruction can execute multiple data in parallel.

SoC *System-on-Chip.* An integrated circuit that integrates all necessary electronic components of a system into a single chip.

Viterbi decoder An algorithm to decode a bitstream that has been encoded using forward error correction based on a convolutional code, developed by Andrew J. Viterbi in 1967.

VLIW *Very Long Instruction Word.* A computer architecture which fetches multiple independent instructions at the same clock cycle to execute them in parallel, to improve the system performance.

WCET *Worst-Case Execution Time.* WCET is the maximum length of time for a task to be executed on a specific hardware platform.

Related publications

1. **Bin Liu** and Bevan Baas, "Parallel AES Encryption Engines for Many-Core Processor Arrays," *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 536–547, March 2013.

2. Brent Bohnenstiehl, Aaron Stillmaker, Jon Pimentel, Timothy Andreas, **Bin Liu**, Anh Tran, Emmanuel Adeagbo and Bevan Baas, "KiloCore: A 32 nm 1000-Processor Computational Array," Submitted to *IEEE Journal of Solid-State Circuits*.

3. **Bin Liu**, Soheil Ghiasi, and Bevan Baas, "Optimizing Power of Many-Core systems by Exploiting Dynamic Voltage Frequency and Core Scaling," journal paper in preparation.

4. **Bin Liu**, Brent Bohnenstiehl, and Bevan Baas, "Scalable Joint Local and Global Dynamic Voltage and Frequency Scaling for Many-Core systems," journal paper in preparation.

5. Brent Bohnenstiehl, Aaron Stillmaker, Jon Pimentel, Timothy Andreas, **Bin Liu**, Anh Tran, Emmanuel Adeagbo and Bevan Baas, "KiloCore: A 32 nm 1000-Processor Array," *IEEE HotChips Symposium on High-Performance Chips, (HotChips)*, August 2016.

6. Brent Bohnenstiehl, Aaron Stillmaker, Jon Pimentel, Timothy Andreas, **Bin Liu**, Anh Tran, Emmanuel Adeagbo and Bevan Baas, "A 5.8 pJ/Op 115 Billion Ops/sec, to 1.78 Trillion Ops/sec 32nm 1000-Processor Array," *IEEE Symposium on VLSI Circuits*, June 2016.

7. **Bin Liu**, Mohammad H. Foroozannejad, Soheil Ghiasi and Bevan Baas, "Optimizing Power of Many-Core systems by Exploiting Dynamic Voltage Frequency and Core Scaling," *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug. 2015.
   Best Student Paper, Third Place.

8. **Bin Liu**, Brent Bohnenstiehl and Bevan Baas, "Scalable Hardware-Based Power Management for Many-Core Systems," *IEEE Asilomar Conference on Signals, Systems and Computers (ACSSC)*, Nov. 2014.

9. **Bin Liu** and Bevan Baas, "Energy-Efficient AES Ciphers on a Fine-Grained Many-Core System," *Technology and Talent for the 21st Century (TECHCON 2012)*, Sept. 2012.

10. **Bin Liu** and Bevan Baas, "A High-Performance Area-Efficient AES Cipher on a Many-Core Platform," *IEEE Asilomar Conference on Signals, Systems and Computers (ACSSC)*, Nov. 2011.

    Nominated for Best Student Paper.

Bibliography

[1] G. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, pp. 114–117, Apr. 1965.

[2] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, pp. 256–268, Oct. 1974.

[3] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick, "The landscape of parallel computing research: A view from berkeley," tech. rep., EECS Department, University of California, Berkeley, Dec. 2006.

[4] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, 1999.

[5] M. Bohr, "A 30 year retrospective on dennard's MOSFET scaling paper," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11–13, 2007.

[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *40th Annual Design Automation Conference*, pp. 338–342, 2003.

[7] R. Dennard, J. Cai, and A. Kumar, "A perspective on today's scaling challenges and possible future directions," *Solid-State Electronics*, vol. 51, pp. 518 – 525, Mar. 2007.

[8] A. Danowitz, K. Kelley, J. Mao, J. Stevenson, and M. Horowitz, "CPU DB: Recording microprocessor history," *ACM Queue - Processors*, vol. 10, pp. 10:10–10:27, Apr. 2012.

[9] P. Gelsinger, "Microprocessors for the new millennium: Challenges, opportunities, and new frontiers," in *Proc. 2001 IEEE International Solid-State Circuits Conference*, pp. 22–25, Feb. 2001.

[10] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang, "A dual-core multi-threaded xeon processor with 16mb l3 cache," in *Proc. 2006 IEEE International Solid-State Circuits Conference*, pp. 315–324, Feb. 2006.

[11] M. Golden, S. Arekapudi, G. Dabney, M. Haertel, S. Hale, L. Herlinger, Y. Kim, K. McGrath, V. Palisetti, and M. Singh, "A 2.6ghz dual-core 64b x86 microprocessor with DDR2 memory support," in *Proc. 2006 IEEE International Solid-State Circuits Conference*, pp. 325–332, Feb. 2006.

[12] J. Hart, S. Choe, L. Cheng, C. Chou, A. Dixit, K. Ho, J. Hsu, K. Lee, and J. Wu, "Implementation of a 4th-generation 1.8ghz dual-core SPARC v9 microprocessor," in *Proc. 2005 IEEE International Solid-State Circuits Conference*, pp. 186–592, Feb. 2005.

[13] J. Friedrich, B. McCredie, N. James, B. Huott, B. Curran, E. Fluhr, G. Mittal, E. Chan, Y. Chan, D. Plass, S. Chu, H. Le, L. Clark, J. Ripley, S. Taylor, J. Dilullo, and M. Lanzerotti, "Design of the power6 microprocessor," in *Proc. 2007 IEEE International Solid-State Circuits Conference*, pp. 96–97, Feb. 2007.

[14] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar, "An integrated quad-core opteron processor," in *Proc. 2007 IEEE International Solid-State Circuits Conference*, pp. 102–103, Feb. 2007.

[15] B. Stackhouse, B. Cherkauer, M. Gowan, P. Gronowski, and C. Lyles, "A 65nm 2-billion-transistor quad-core itanium processor," in *Proc. 2008 IEEE International Solid-State Circuits Conference*, pp. 92–598, Feb. 2008.

[16] R. Kuppuswamy, S. R. Sawant, S. Balasubramanian, P. Kaushik, N. Natarajan, and J. D. Gilbert, "Over one million TPCC with a 45nm 6-core xeon cpu," in *Proc. 2009 IEEE International Solid-State Circuits Conference*, pp. 70–71, Feb. 2009.

[17] U. G. Nawathe, M. Hassan, L. Warriner, K. Yen, B. Upputuri, D. Greenhill, A. Kumar, and H. Park, "An 8-core 64-thread 64b power-efficient SPARC soc," in *Proc. 2007 IEEE International Solid-State Circuits Conference*, pp. 108–590, Feb. 2007.

[18] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," Aug. 2009. `http://www.gotw.ca/publications/concurrency-ddj.htm`.

[19] M. Hill and M. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, pp. 33–38, July 2008.

[20] H. Esmaeilzadeh, E. Blem, R. Amant, K. Sankaralingam, and D. Burger, "Power challenges may end the multicore era," *Commun. ACM*, vol. 56, pp. 93–102, Feb. 2013.

[21] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annual International Symposium on Computer Architecture*, pp. 365–376, June 2011.

[22] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C. C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64 - processor: A 64-core soc with mesh interconnect," in *Proc. 2008 IEEE International Solid-State Circuits Conference*, pp. 88–598, Feb. 2008.

[23] E. Semiconductor, "TILE-Gx72 processor product brief," Feb. 2015. `http://www.tilera.com/files/drim__TILE-Gx8072_PB041-04_WEB_7666.pdf`.

[24] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-w teraflops processor in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 43, pp. 29–41, Jan. 2008.

[25] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. V. D. Wijngaart, "A 48-core ia-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling," *IEEE Journal of Solid-State Circuits*, vol. 46, pp. 173–183, Jan. 2011.

[26] B. de Dinechin, R. Ayrignac, P. Beaucamps, P. Couvert, B. Ganne, P. de Massas, F. Jacquet, S. Jones, N. Chaisemartin, F. Riss, and T. Strudel, "A clustered manycore processor architecture for embedded and accelerated applications," in *Proc. 2013 IEEE High Performance Extreme Computing Conference*, pp. 1–6, 2013.

[27] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, D. Truong, T. Mohsenin, and B. Baas, "AsAP: An asynchronous array of simple processors," *IEEE Journal of Solid-State Circuits*, vol. 43, pp. 695–705, Mar. 2008.

[28] D. Truong, W. Cheng, T. Mohsenin, Z. Y. T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, P. Mejia, A. Tran, J. Webb, E. Work, Z. Xiao, and B. Baas, "A 167-processor computational array for highly-efficient DSP and embedded application processing," in *Proc. 2008 IEEE HotChips Symposium on High-Performance Chips*, Aug. 2008.

[29] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. M. Baas, "A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array," in *Proc. IEEE Symp. VLSI Circuits*, June 2016.

[30] M. Meeuwsen, O. Sattari, and B. Baas, "A full-rate software implementation of an IEEE 802.11a compliant digital baseband transmitter," in *Proc. 2004 IEEE Workshop on Signal Processing Systems*, Oct. 2004.

[31] A. Tran, D. Truong, and B. Baas, "A complete real-time 802.11a baseband receiver implemented on an array of programmable processors," in *Proc. 2008 IEEE Asilomar Conference on Signals, Systems and Computers*, pp. 165–170, Oct. 2008.

[32] A. Tran, D. Truong, and B. Baas, "A GALS many-core heterogeneous DSP platform with source-synchronous on-chip interconnection network," in *Proc. 3rd ACM/IEEE International Symposium on Networks-on-Chip*, May. 2009.

[33] Z. Xiao and B. Baas, "A high-performance parallel CAVLC encoder on a fine-grained many-core system," in *Proc. 2008 International Conference on Computer Design*, pp. 248–254, Oct. 2008.

[34] S. Le, "A fine grained many-core h.264 video encoder," Master's thesis, University of California, Davis, CA, USA, Mar. 2010. `http://www.ece.ucdavis.edu/vcl/pubs/theses/2010-03`.

[35] Z. Xiao and B. Baas, "A 1080p H.264/AVC baseline residual encoder for a fine-grained many-core system," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, pp. 890–902, july 2011.

[36] Z. Xiao, S. Le, and B. Baas, "A fine-grained parallel implementation of a H.264/AVC encoder on a 167-processor computational platform," in *Proc. 2011 IEEE Asilomar Conference on Signals, Systems and Computers*, Nov. 2011.

[37] D. Truong and B. Baas, "Massively parallel processor array for mid-/back-end ultrasound signal processing," in *Proc. 2010 IEEE Biomedical Circuits and Systems Conference*, Nov. 2010.

[38] B. Liu and B. Baas, "A high-performance area-efficient AES cipher on a many-core platform," in *Proc. 45th Asilomar Conference on Signals, Systems and Computers*, pp. 2058–2062, Nov. 2011.

[39] B. Liu and B. Baas, "Energy-efficient AES ciphers on a fine-grained many-core system," in *Proc. 2012 Technology and Talent for the 21st Century*, Sept. 2011.

[40] B. Liu and B. Baas, "Parallel AES encryption engines for many-core processor arrays," *IEEE Transactions on Computers*, vol. 62, pp. 536–547, Mar. 2013.

[41] A. Stillmaker, L. Stillmaker, and B. Baas, "Fine-grained energy-efficient sorting on a many-core processor array," in *Proc. 18th IEEE International Conference on Parallel and Distributed Systems*, pp. 652–659, dec. 2012.

[42] B. Bohnenstiehl and B. Baas, "A software LDPC decoder implemented on a many-core array of programmable processors," in *Proc. 49th Asilomar Conference on Signals, Systems and Computers*, pp. 192–196, Nov. 2015.

[43] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Intergrated Circuits – A Design Perspective*. New Jersey, NJ: Prentice-Hall, second ed., 2003.

[44] H. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *IEEE Journal of Solid-State Circuits*, vol. 19, pp. 468–473, Aug. 1984.

[45] S. Rusu, "Power and leakage reduction in the nanoscale era," Aug. 2008. `http://www.ewh.ieee.org/r6/scv/ssc/Aug08.pdf`.

[46] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE*, vol. 91, pp. 305–327, Feb. 2003.

[47] N. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, pp. 68–75, Dec. 2003.

[48] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, pp. 415–420, Mar. 2000.

[49] E. Friedman, "Clock distribution networks in synchronous digital integrated circuits," *Proceedings of the IEEE*, vol. 89, pp. 665–692, May 2001.

[50] J. Shin, Y.and Seomun, K. Choi, and T. Sakurai, "Power gating: Circuits, design methodologies, and best practice for standard-cell VLSI designs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 15, pp. 28:1–28:37, Oct. 2010.

[51] H. Jiang, M. Marek-Sadowska, and S. Nassif, "Benefits and costs of power-gating technique," in *Proc. 2005 IEEE International Conference on Computer Design*, pp. 559–566, Oct. 2005.

[52] S. Kim, S. Kosonocky, and D. Knebel, "Understanding and minimizing ground bounce during mode transition of power gating structures," in *Proc. 2003 ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 22–25, Aug. 2003.

[53] K. Usami and N. Ohkubo, "A design approach for fine-grained run-time power gating using locally extracted sleep signals," in *Proc. 2006 IEEE International Conference on Computer Design*, pp. 155–161, Oct. 2006.

[54] J. Lee and N. Kim, "Analyzing potential throughput improvement of power- and thermal-constrained multicore processors by exploiting DVFS and PCPG," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 20, pp. 225–235, Feb. 2012.

[55] L. Wei, Z. Chen, M. Johnson, K. Roy, and V. De, "Design and optimization of low voltage high performance dual threshold cmos circuits," in *Proc. 35th Annual Design Automation Conference*, pp. 489–494, June 1998.

[56] K. Usami, N. Kawabe, M. Koizumi, K. Seta, and T. Furusawa, "Automated selective multi-threshold design for ultra-low standby applications," in *Proc. 2002 ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 202–206, Aug. 2002.

[57] K. Nose and T. Sakurai, "Optimization of vdd and vth for low-power and high-speed applications," in *Proc. 2000. ACM/IEEE Asia and South Pacific Design Automation Conference*, pp. 469–474, June 2000.

[58] A. Srivastava and D. Sylvester, "Minimizing total power by simultaneous vdd/vth assignment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, pp. 665–677, May 2004.

[59] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson, and K. Keutzer, "Minimization of dynamic and static power through joint assignment of threshold voltages and sizing optimization," in *Proc. 2003 ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 158–163, Aug. 2003.

[60] A. Keshavarzi, S. Narendra, S. Borkar, C. Hawkins, K. Roy, and V. De, "Technology scaling behavior of optimum reverse body bias for standby leakage power reduction in CMOS IC's," in *Proc. 1999 ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 252–254, Aug. 1999.

[61] L. Clark, M. Morrow, and W. Brown, "Reverse-body bias and supply collapse for low effective standby power," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 12, no. 9, pp. 947–956, 2004.

[62] A. Keshavarzi, S. Ma, S. Narendra, B. Bloechel, K. Mistry, T. Ghani, S. Borkar, and V. De, "Effectiveness of reverse body bias for leakage control in scaled dual vt CMOS ICs," in *Proc. 2001 ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 207–212, Aug. 2001.

[63] S. Narendra, D. Antoniadis, and V. De, "Impact of using adaptive body bias to compensate die-to-die vt variation on within-die vt variation," in *Proc. 1999 ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 229–232, Aug. 1999.

[64] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De, "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage," *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 1396–1402, Nov. 2002.

[65] T. Chen and S. Naffziger, "Comparison of adaptive body bias (ABB) and adaptive supply voltage (ASV) for improving delay and leakage under the presence of process variation," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 11, pp. 888–899, Oct. 2003.

[66] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473–484, Apr. 1992.

[67] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: an approach for energy efficient computing," in *Proc. 1996 IEEE International Symposium on Low-Power Electronics and Design*, pp. 347–352, Aug. 1996.

[68] A. Sinha and A. Chandrakasan, "Dynamic voltage scheduling using adaptive filtering of workload traces," in *Proc. 14th IEEE International Conference on VLSI Design*, pp. 221–226, Jan. 2001.

[69] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, pp. 584–600, May 2004.

[70] D. Zhu, R. Melhem, and B. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, pp. 686–700, July 2003.

[71] Q. Wu, P. Juang, M. Martonosi, and D. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors," in *Proc. 11th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 248–259, Dec. 2004.

[72] A. Alimonda, S. Carta, A. Acquaviva, A. Pisano, and L. Benini, "A feedback-based approach to DVFS in data-flow applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, pp. 1691–1704, Nov. 2009.

[73] P. Choudhary and D. Marculescu, "Hardware based frequency/voltage control of voltage frequency island systems," in *Proc. 4th IEEE International Conference Hardware/Software Codesign and System Synthesis*, pp. 34–39, Oct. 2006.

[74] B. Liu, B. Bohnenstiehl, and B. Baas, "Scalable hardware-based power management for many-core systems," in *Proc. 48th IEEE Asilomar Conference on Signals, Systems and Computers*, pp. 1834–1838, Nov. 2014.

[75] D. Marculescu, "On the use of microarchitecture-driven dynamic voltage scaling," 2000.

[76] S. Ghiasi, J. Casmira, and D. Grunwald, "Using IPC variation in workloads with externally specified rates to reduce power consumption," in *In Workshop on Complexity Effective Design*, 2000.

[77] G. Dhiman and T. Rosing, "Dynamic voltage frequency scaling for multi-tasking systems using online learning," in *Proc. 2007 ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 207–212, Aug. 2007.

[78] B. Zhai, R. Dreslinski, D. Blaauw, T. Mudge, and D. Sylvester, "Energy efficient near-threshold chip multi-processing," in *Proc. 2007 ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 32–37, Aug. 2007.

[79] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, pp. 253–266, Feb. 2010.

[80] B. Zhai, D. Blaauw, D. Sylvester, and S. Hanson, "A sub-200mv 6t SRAM in 0.13μm CMOS," in *Proc. 2007 IEEE International Solid-State Circuits Conference*, pp. 332–606, Feb. 2007.

[81] B. Zhai, S. Pant, L. Nazhandali, S. Hanson, J. Olson, A. Reeves, M. Minuth, R. Helfand, T. Austin, D. Sylvester, and D. Blaauw, "Energy-efficient subthreshold processor design," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 17, pp. 1127–1137, Aug. 2009.

[82] J. Li and J. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," in *Proc. 12th IEEE International Symposium on High Performance Computer Architecture*, pp. 77–87, Feb. 2006.

[83] J. Lee, V. Sathisha, M. Schulte, K. Compton, and N. Kim, "Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling," in *Proc. 20th IEEE International Conference on Parallel Architectures and Compilation Techniques*, pp. 111–120, Oct. 2011.

[84] B. Liu, M. Foroozannejad, S. Ghiasi, and B. Baas, "Optimizing power of many-core systems by exploiting dynamic voltage, frequency and core scaling," in *Proc. 58th IEEE International Midwest Symposium on Circuits and Systems*, Aug. 2015.

[85] F. Pollack, "New microarchitecture challenges in the coming generations of CMOS process technologies," in *Proc. 32nd Annual ACM/IEEE International Symposium on Microarchitecture*, Nov. 1999.

[86] S. Borkar, "Thousand core chips: a technology perspective," in *Proc. 44th Annual Design Automation Conference*, pp. 746–749, June 2007.

[87] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating MapReduce for multi-core and multiprocessor systems," in *Proc. 13th IEEE International Symposium on High Performance Computer Architecture*, pp. 13–24, Feb. 2007.

[88] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, P. Mejia, A. Tran, J. Webb, E. Work, Z. Xiao, and B. Baas, "A 167-processor 65 nm computational platform with per-processor dynamic supply voltage and dynamic clock frequency scaling," in *Proc. 2008 IEEE Symposium on VLSI Circuits*, pp. 22–23, June 2008.

[89] A. Tran, D. Truong, and B. Baas, "A reconfigurable source-synchronous on-chip network for GALS many-core platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, pp. 897–910, June 2010.

[90] R. Apperson, Z. Yu, M. Meeuwsen, T. Mohsenin, and B. Baas, "A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 15, pp. 1125–1134, Oct. 2007.

[91] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, A. Jacobson, G. Landge, M. Meeuwsen, A. Tran, Z. Xiao, E. Work, J. Webb, P. Mejia, and B. Baas, "A 167-processor computational platform in 65 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 44, pp. 1130–1144, Apr. 2009.

[92] R. Jevtic, H. P. Le, M. Blagojevic, S. Bailey, K. Asanovic, E. Alon, and B. Nikolic, "Per-core DVFS with switched-capacitor converters for energy efficiency in manycore processors," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 23, pp. 723–730, Apr. 2015.

[93] M. Hashemi, M. Foroozannejad, S. Ghiasi, and C. Etzel, "FORMLESS: Scalable utilization of embedded manycores in streaming applications," pp. 71–78, May 2012.

[94] B. Baas, Z. Yu, M. Meeuwsen, O. Sattari, R. Apperson, E. Work, J. Webb, M. Lai, T. Mohsenin, D. Truong, and J. Cheung, "AsAP: A fine-grained many-core platform for DSP applications," *IEEE Micro*, vol. 27, pp. 34–45, Mar. 2007.

[95] E. W. Work, "Algorithms and software tools for mapping arbitrarily connected tasks onto an asynchronous array of simple processors," Master's thesis, University of California, Davis, CA, USA, Sept. 2007.

[96] NIST, "Advanced encryption standard (AES)," Nov. 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[97] NIST, "Data encryption standard (DES)," Oct. 1999. `http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`.

[98] I. Verbauwhede, P. Schaumont, and H. Kuo, "Design and performance testing of a 2.29 gb/s rijndael processor," *IEEE Journal of Solid-State Circuits*, vol. 38, pp. 569–572, Mar. 2003.

[99] D. Mukhopadhyay and D. Roychowdhury, "An efficient end to end design of rijndael cryptosystem in 0.18 μm CMOS," in *Proc. 18th IEEE International Conference on VLSI Design*, pp. 405–410, Jan. 2005.

[100] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. San Francisco, CA: Morgan Kaufmann, fourth ed., 2007.

[101] S. Morioka and A. Satoh, "A 10-gbps full-AES crypto design with a twisted BDD s-box architecture," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 12, pp. 686–691, July 2004.

[102] J. Daemen and V. Rijmen, *The Design of Rijndael*. New York, NY: Springer-Verlag, 2002.

[103] A. Hodjat and I. Verbauwhede, "Area-throughput trade-offs for fully pipelined 30 to 70 gbits/s AES processors," *IEEE Transactions on Computers*, vol. 55, pp. 366–372, Apr. 2006.

[104] S. Mathew, F. Sheikh, M. Kounavis, S. Gueron, A. Agarwal, S. Hsu, H. Kaul, M. Anders, and R. Krishnamurthy, "53 gbps native GF($(2^4)^2$) composite-field AES-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 46, pp. 767–776, Apr. 2011.

[105] A. Hodjat and I. Verbauwhede, "A 21.54 gbits/s fully pipelined AES processor on FPGA," in *Proc. 12th IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 308–309, Apr. 2004.

[106] C.-J. Chang, C.-W. Huang, K.-H. Chang, Y.-C. Chen, and C.-C. Hsieh, "High throughput 32-bit AES implementation in FPGA," in *IEEE Asia Pacific Conference on Circuits and Systems*, pp. 1806–1809, Nov. 2008.

[107] J. Granado-Criado, M. Vega-Rodriguez, J. Sanchez-Perez, and J. Gomez-Pulido, "A new methodology to implement the AES algorithm using partial and dynamic reconfiguration," *Integration, the VLSI Journal*, vol. 43, no. 1, pp. 72–80, 2010.

[108] S. Qu, G. Shou, Y. Hu, Z. Guo, and Z. Qian, "High throughput, pipelined implementation of AES on FPGA," in *Proc. 2009 IEEE International Symposium on Information Engineering and Electronic Commerce*, pp. 542–545, May 2009.

[109] "International technology roadmap for semiconductors, design," 2009. `http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_Design.pdf`.

[110] M. Matsui and J. Nakajima, "On the power of bitslice implementation on intel core2 processor," in *Cryptographic Hardware and Embedded Systems*, vol. 4727 of *Lecture Notes in Computer Science*, pp. 121–134, 2007.

[111] E. Biham, "A fast new DES implementation in software," in *Fast Software Encryption*, vol. 1267 of *Lecture Notes in Computer Science*, pp. 260–272, 1997.

[112] D. Bernstein and P. Schwabe, "New AES software speed records," in *Progress in Cryptology*, vol. 5365 of *Lecture Notes in Computer Science*, pp. 322–336, 2008.

[113] "Supplemental streaming SIMD extensions 3." `http://en.wikipedia.org/wiki/SSSE3`.

[114] E. Kasper and P. Schwabe, "Faster and timing-attack resistant AES-GCM," in *Cryptographic Hardware and Embedded Systems*, vol. 5747 of *Lecture Notes in Computer Science*, pp. 1–17, 2009.

[115] S. Gueron, "Intel advanced encryption standard (AES) instructions set," Jan. 2010.

[116] T. Wollinger, M. Wang, J. Cuajardo, and C. Paar, "How well are high-end DSPs suited for the AES algorithm?," in *The Third AES Candidate Conference*, pp. 94–105, Apr. 2000.

[117] S. Manavski, "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography," in *Proc. 2007 IEEE International Conference on Signal Processing and Communications*, pp. 65–68, Nov. 2007.

[118] Z. Yu and B. Baas, "A low-area multi-link interconnect architecture for GALS chip multiprocessors," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 18, pp. 750–762, May 2010.

[119] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm early design exploration," *IEEE Transactions on Electron Devices*, vol. 53, pp. 2816–2823, Nov. 2006.

[120] A. Stillmaker, "Exploration of technology scaling of CMOS circuits from 180 nm to 22 nm using PTM models in HSPICE," *Technical Report UC Davis*, June 2011.

[121] S. Mathew, S. Satpathy, V. Suresh, M. Anders, H. Kaul, A. Agarwal, S. Hsu, G. Chen, and R. Krishnamurthy, "340 mv - 1.1 v, 289 gbps/w, 2090-gate nano AES hardware accelerator with area-optimized encrypt/decrypt GF($(2^4)^2$) polynomials in 22 nm tri-gate CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, pp. 1048–1058, Apr. 2015.

[122] Y. Zhang, K. Yang, M. Saligane, D. Blaauw, and D. Sylvester, "A compact 446 gbps/w AES accelerator for mobile soc and iot in 40nm," in *2016 IEEE Symposium on VLSI Circuits*, pp. 1–2, June 2016.

[123] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. M. Baas, "Kilocore: A 32 nm 1000-processor array," in *Proc. IEEE HotChips Symposium on High-Performance Chips (HotChips)*, Aug. 2016.

[124] W. Cheng and B. Baas, "Dynamic voltage and frequency scaling circuits with two supply voltages," in *Proc. 2008 IEEE International Symposium on Circuits and Systems*, pp. 1236–1239, May 2008.

[125] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *Proc. 14th International Symposium on High Performance Computer Architecture*, pp. 123–134, Feb. 2008.

[126] D. Mosse, H. Aydin, B. Childers, and R. Melhem, "Compiler-assisted dynamic power-aware scheduling for real-time applications," in *Workshop on Compilers and Operating Systems for Low Power*, 2000.

[127] D. Zhu, D. Mosse, and R. Melhem, "Power-aware scheduling for and/or graphs in real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, pp. 849–864, Sept. 2004.

[128] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 18–28, Jan. 2005.

[129] U. Ogras, R. Marculescu, and D. Marculescu, "Variation-adaptive feedback control for networks-on-chip with multiple clock domains," in *Proc. 45th Annual Design Automation Conference*, pp. 614–619, June 2008.

[130] S. Garg, D. Marculescu, and R. Marculescu, "Custom feedback control: Enabling truly scalable on-chip power management for MPSoCs," in *Proc. 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design*, pp. 425–430, Aug. 2010.

[131] J. Lee and N. Kim, "Optimizing throughput of power- and thermal-constrained multicore processors using DVFS and per-core power-gating," in *Proc. 46th Annual Design Automation Conference*, pp. 47–50, July 2009.

[132] V. Gutnik and A. Chandrakasan, "Embedded power supply for low-power dsp," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 5, no. 4, pp. 425–435, 1997.

[133] W. Cheng, "Approaches and designs of dynamic voltage and frequency scaling," Master's thesis, University of California, Davis, CA, USA, Jan. 2008.

[134] Z. Yu, *High Performance and Energy Efficient Multi-core Systems for DSP Applications*. PhD thesis, University of California, Davis, CA, USA, Oct. 2007.

[135] S. Borkar and A. Chien, "The future of microprocessors," *Communications of the ACM*, pp. 67–77, 2011.

[136] L. Wei, Z. Chen, K. Roy, M. Johnson, Y. Ye, and V. De, "Design and optimization of dual-threshold circuits for low-voltage low-power applications," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 7, pp. 16–24, Mar. 1999.

[137] D. Goldberg, "Genetic algorithms in search, optimization, and machine learning," 1989.

[138] M. Butler, "AMD Bulldozer Core - a new approach to multithreaded compute performance for maximum efficiency and throughput," in *Proc. IEEE HotChips Symposium on High-Performance Chips (HotChips)*, Aug. 2010.