DESIGN METHODOLOGIES FOR RELIABLE AND ENERGY EFFICIENT MULTIPROCESSOR SYSTEM

ANUP KUMAR DAS

NATIONAL UNIVERSITY OF SINGAPORE

2014

DESIGN METHODOLOGIES FOR RELIABLE AND ENERGY EFFICIENT MULTIPROCESSOR SYSTEM

ANUP KUMAR DAS

(B.Eng. (First Class, Hons.), Jadavpur University, India)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2014

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not being submitted for any degree in any university previously.

Anup Kumar Das
15 May 2014

Acknowledgments

I have always regarded the PhD journey as a platform for developing intelligence, creativity, curiosity, adaptability, self-motivation, competitiveness, and above all, maturity. As such, I have valued this journey more than the destination itself. Spending three years in this journey has been a fulfilling experience for me. It is difficult to say whether this is due to the research topic itself, which touched my heart, or the joy of paper acceptance at regular intervals, which boosted my moral, or the fruitful discussions with numerous people who accompanied me in this journey. In any case, I am indebted to all those, who have helped me to succeed in reaching the destination.

First of all, I would like to express my heartfelt gratitude to Dr. Akash Kumar, my supervisor all through the last three years. It has been an honor to be his first PhD student at the NUS. In him, I found a great mentor and a lifetime friend. He taught me to become a mature researcher, helping me push my boundaries and work on projects beyond my comfort zone. His enthusiasm for research and his supportiveness have been a continuous source of motivation, even during tough times in this journey. I appreciate all his time that he devoted for me, both for research discussion and non-academic activities, such as sports and travel. His eyes for perfection have been a benchmark for me all through these years. I couldnt have asked for a better role model than him, and I hope to promote the research values he has given me, to all those whom I come across during the academic life beyond PhD.

I would also like to thank Prof. Bharadwaj Veeravalli, for co supervising me during

the last three years. He has been supportive all through my PhD duration. He gave me useful insights about the fault-tolerant research and provided critical comments on my publications. He has helped me to formulate and solve optimization problems that were well received and fetched lot of praises among research communities. He also helped me with the administrative things at NUS. I was very fortunate to have two supervisors, who were highly motivating and supportive.

In addition, I have been very privileged to get to know and to collaborate with some of the great research groups around the world. I would especially like to thank Prof. Bashir M. Al-Hashimi, Dr. Geoff V. Merrett and Dr. Rishad A. Shafik from the University of Southampton, who hosted me in their research group for six months and have given me the opportunity to work on one of the most interesting projects of my PhD duration that I intend to carry forward with, to the next stage of my academic career. I am also thankful to Prof. Cristiana Bolchini and Dr. Antonio Miele, who have welcomed me in their research group at the Politecnico Di Milano for a period of three months. During this period I was fortunate to be part of some fruitful discussions about reliability that helped me to answer some of my own research problems.

For the last three years I had the pleasure to work in the VLSI Lab at NUS. I would like to thank all my group members, especially Dr. Amit Kumar Singh, Shyam, Li Ang and Pham for making my stay memorable. I enjoyed the friendly atmosphere and the discussions we had during lunch and coffee breaks. My time at Southampton was made enjoyable in large part due to the many friends that became part of my life. I am grateful to Domenico, Matt, Jedrzej (Andy), Tual, Alessandra and Julia for the memorable trips together and numerous international lunches and dinners. My thanks also extend to Matteo Carminati from Politecnico Di Milano, who visited Singapore during the last phase of my PhD and with whom I spent good times discussing non-research related topics, which helped me to ease the stress of thesis writing.

I would like to thank my parents and my sister for all their love and encouragement to make this journey an enriching one. Last but not least, I would like to thank my best friend, and a loving, supportive, and encouraging wife, Sudipta, who has helped me to embark on this long journey and accompany me till the destination.

Anup Kumar Das

Contents

Summary

As the performance demands of applications (e.g., multimedia) are growing, multiple processing cores are integrated together to form multiprocessor systems. Energy minimization is a primary optimization objective for these systems. An emerging concern for designs at deep-submicron technology nodes (65nm and below) is the lifetime reliability, as escalating power density and hence temperature variation continues to accelerate wear-out leading to a growing prominence of device defects. As such, reliability and energy need to be incorporated in the multiprocessor design methodology, addressing two key aspects:

- lifetime amelioration, i.e. improving the lifetime reliability through energy- and performance-aware intelligent task mapping; and
- graceful degradation, i.e. determining the task mapping for different fault-scenarios while minimizing the energy consumption and providing graceful performance degradation.

In this thesis, a platform-based design methodology is first proposed to minimize temperature-related wear-outs. Fundamental to this methodology is a temperature model that predicts the temperature of a core incorporating not only its dependency on the voltage and frequency of operation (temporal effect), but also its dependency on the temperature of the surrounding cores (spatial effect). The proposed temperature model is integrated in a gradient-based fast heuristic that controls the voltage and frequency of the cores to limit the average and peak temperature leading to a longer lifetime, simultaneously minimizing the energy consumption.

A design flow is then proposed as a part of the hardware-software co-design methodology to determine the minimum number of cores and the size of the FPGA fabric of a reconfigurable multiprocessor system. The objective is to maximize the lifetime reliability of the cores while satisfying a given area, performance, and energy budget. The proposed flow incorporates individual as well concurrent applications with different performance requirements and thermal behaviors. While the existing studies determine platform architecture for energy and area minimization, this is the first approach for reconfigurable multiprocessor system design considering lifetime reliability together with multi-application use-cases.

To provide graceful performance degradation in the presence of faults, a reactive fault-tolerance technique is also proposed that explores different task mapping alternatives to minimize energy consumption while guaranteeing throughput for all processor fault-scenarios. Directed acyclic graphs (DAGs) and synchronous data flow graphs (SDFGs) are used to model applications making the proposed methodology applicable to streaming multimedia and non-multimedia applications. Fundamental to this approach is a novel scheduling algorithm based on self-timed execution, which minimizes both the schedule storage overhead and run-time schedule construction overhead. Unlike the existing approaches which consider task mapping only, the proposed technique considers task mapping and scheduling in an integrated manner, achieving significant improvement with respect to these state-of-the-art approaches.

Finally, an adaptive run-time manager is designed for lifetime amelioration of multiprocessor systems by managing thermal variations, both within (intra) and across (inter) applications. Core to this approach is a reinforcement-learning algorithm, which interfaces with the on-board thermal sensors and controls the voltage and frequency of operation and the thread-to-core affinity over time. This approach is built on top of the design-time analysis to minimize learning time, address run-time variability and support new applications, making the overall thesis objective to provide a complete and systematic design solution for reliable and energy-efficient multiprocessor systems.

List of Tables

List of Figures

CHAPTER 1

Trends in Multiprocessor Systems Design

1.1 Trends in Transistor Scaling

Technology scaling principles introduced in 1974 [4] have revolutionized the semiconductor industry, providing a roadmap for systematic and predictable improvement in transistor density, switching speed and power dissipation, with scaled transistor feature size. Ever since these principles were introduced, the transistor feature size reduced by $\sqrt{2}$ approximately every 18 months, starting from a CMOS gate length of 1mm in 1974 [4] to 22nm in 2012 [5]. Many technology barriers were perceived along this path and scientists responded with innovations to circumvent, surmount and tunnel through these challenges. Some of these innovations include the use of Strained-Silicon for 90nm CMOS [6], High-κ/Metal Gate for 32nm CMOS [7] and the Tri-Gate technology for 22nm CMOS [5]. However, post 22nm CMOS gate length offered severe challenges to the designers due to pronounced transistor short-channel effects, threatening the decline of Dennard's Scaling principles and confounding the International Technology Roadmap for Semiconductors (ITRS) [8]. New research directions were sought and FinFET technology was adopted due to its superior short channel effects, achieving a gate length as low as 7nm in 2013 [9]. As the technology scaled approximately every two years, the transistor integration capacity doubled. This enabled semiconductor manufacturers to

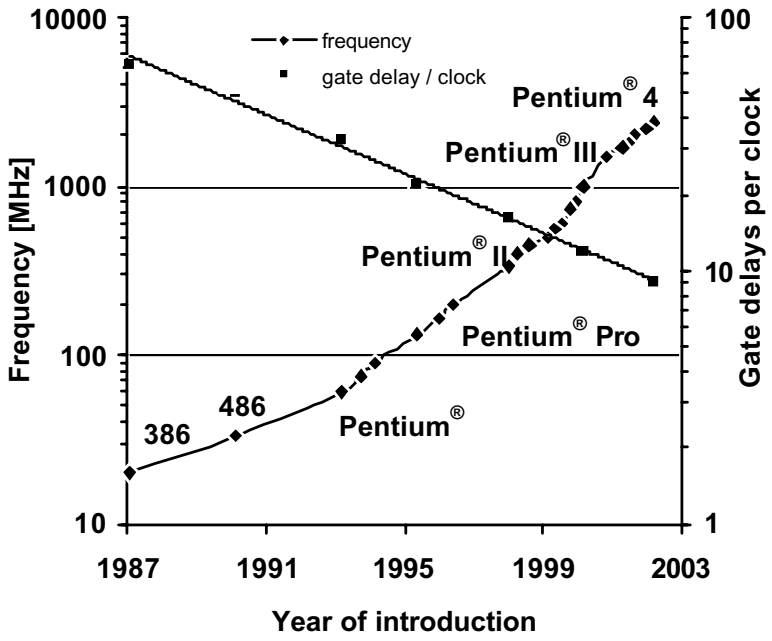


Figure 1.1: Trend in microprocessor development (source: Intel [1]).

pack millions of logic gates per chip.

1.2 Trends in Microprocessor Design

Just as the transistor technology has evolved all these years, so has the microprocessor technology since the first microprocessor was introduced in 1971 by Intel. This first microprocessor used 2300 transistors and was based on 4-bit computation. Shortly afterwards, Intel developed the 8-bit microprocessor family – 8008 and 8080 in 1972 and 1974, respectively. Thereafter, the microprocessor technology advanced to 16-bit (Intel 8086 in 1978), 32-bit (Intel 80386 in 1985) and finally to the Pentium series starting from 1993. Although, the microprocessor technhology was pioneered by Intel, the journey saw several other key players such as Motorola, AMD and ARM, entering into the microprocessor market. Each generation of the processors grew smaller and faster. This growth was guided mainly by the observation of Gordon Moore (co-founder of Intel), known as the Moore's Law, which states that the computer performance will double every 18 months. Thus, Moore's Law together with Dannard's Scaling principle enabled building faster and sophisticated microprocessors.

The trend in the microprocessor development is shown in Figure 1.1. Throughout the 1990's and early 2000, microprocessor frequency was synonymous with performance; higher supported frequency meant a higher performance. This notion of performance was

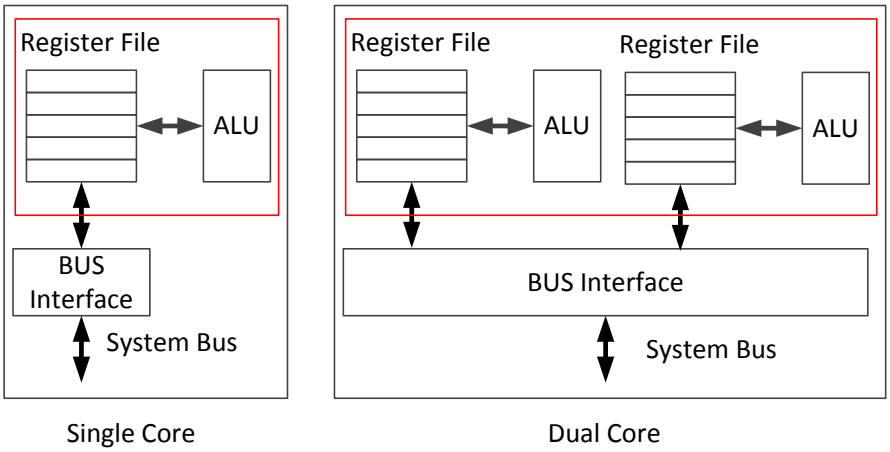


Figure 1.2: Multi-core processor.

soon adjusted to consider other aspects, such as temperature hotspots, power dissipation and energy consumption. In perspective of these new parameters, performance was predicted to deviate significantly from the trend set by Moore's Law. This is because, with every new microprocessor generation, physical dimensions of chips decreased while the number of transistors per chip increased; the maximum clock frequency was thus bounded by a power envelop, crossing which could potentially burn the chip. Many techniques were sought to improve the performance of microprocessor without upscaling the frequency. Some of these techniques included out-of-order execution, pipelining, multithreading, use of reorder buffers and data value predictions.

However, a paradigm shift was inevitable to push forward the performance boundary. One of the major breakthroughs of the early 2000 was the concept of **multi-core** processor – pioneered by research teams from IBM and Intel. A multi-core processor refers to a single computing unit with two or more microprocessor components (such as arithmetic and logic unit, ALU) such that multiple instructions can be read and executed in parallel. These parallel components are referred to as cores of a multi-core processor. Figure 1.2 shows a simplified representation of a single-core and a dual-core processor. The register file and the ALU of the single-core (shown in the red box) are replicated to form the dual-core processor. The other components, such as the bus interface and the system bus, are shared across the two cores. In computer terminology, the term "processor" is often used as an umbrella term to refer to both single core computing unit (microprocessor) or multi-core computing unit (multi-core processor).

Multi-core processors addressed the thermal, power and energy concerns. A processor with two cores running at a clock speed of f, can process the same number of instructions

Table 1.1: State-of-the-art multi- and many-core processors.

Manufacturer	# Cores	Release Year	Technology Node	Max. Power	Max. Frequency
Multi-core Processors					
IBM POWER7	8	2010	45nm	–	4.25 GHz
AMD Bulldozer	4	2011	32nm	125W	3.9 GHz
Intel i7 Haswell	4	2013	22nm	84W	3.5GHz
Many-core Processors					
Cell BE	9	2005	65nm	100W	3.2 GHz
Oracle SPARC M6	12	2013	28nm	–	3.6 GHz
Intel Xeon Phi	60	2012	22nm	225W	1.05 GHz

as that processed by a single core processor running at $2f$ within the same time interval; yet the dual core processor would still consume less energy. Furthermore, the power dissipation is distributed and so do thermal hotspots. This has motivated researchers over the next few years to integrate more cores in a single processor – 2-cores (Intel Core 2 Duo, 2006), 4-cores (Intel Core i5, 2009), 6-cores (Intel Core i7, 2010) and 8-cores (Intel Xeon 2820, 2011). As the technology continued to scale further, the processor architecture started shifting from multi-core to many-core. Although there is no clear consensus among researchers to classify a processor as multi- or many-core based on the number of comprising cores, we adopt one of the popular beliefs, classifying a processor with more than 8 cores as many-core processor. Table 1.1 reports the state-of-the-art multi- and many-core processors from some of the common microprocessor manufacturers.

1.3 Evolution of Multiprocessor Systems

As the semiconductor fabrication technology matured with ever new technology nodes, there emerged a strong demand, especially from the consumer market sector such as mobile phones, towards single chip computing. This new area of innovation can be dated back to 1990 and was coined as **System-on-Chip** (SoC). An SoC is often defined as a single chip complex integrated circuit, incorporating the major processing elements of a complete end-product. Usually, an SoC comprises of a single full-fledged processor (acting as the master) to coordinate the operation of the other processing elements of the system. The earlier SoCs included a single-core processor (i.e., microprocessor). The SheevaPlug SoC [10] with one Kirkwood 6281 processor from Marvell Semiconductors is an example of a single-core SoC. As the processor technology shifted from single-core towards multi-core, the SoC designers started integrating multi-core processors for faster

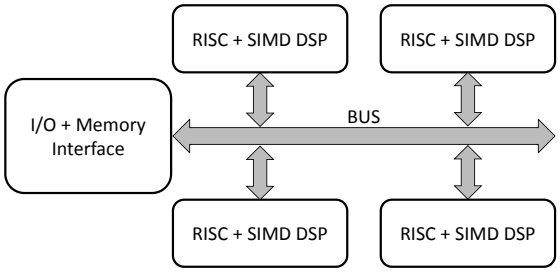


Figure 1.3: Lucent's Daytona architecture [2].

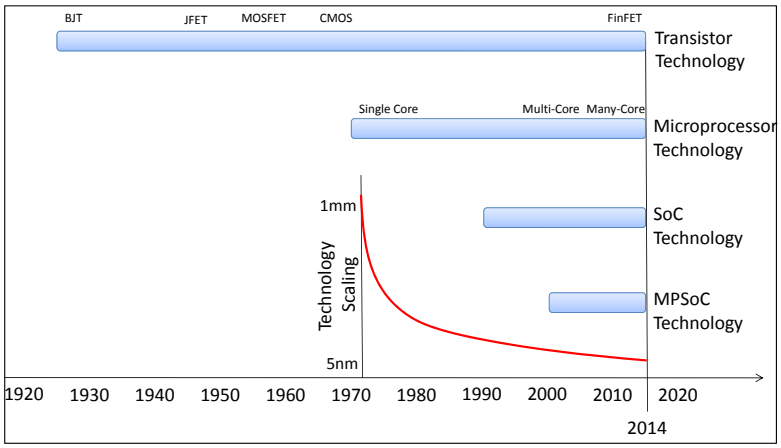
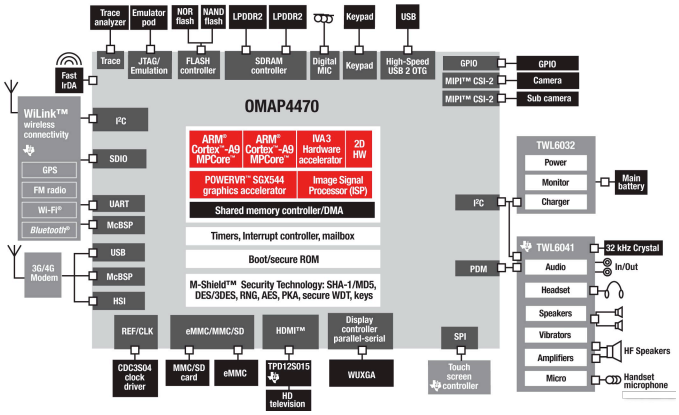


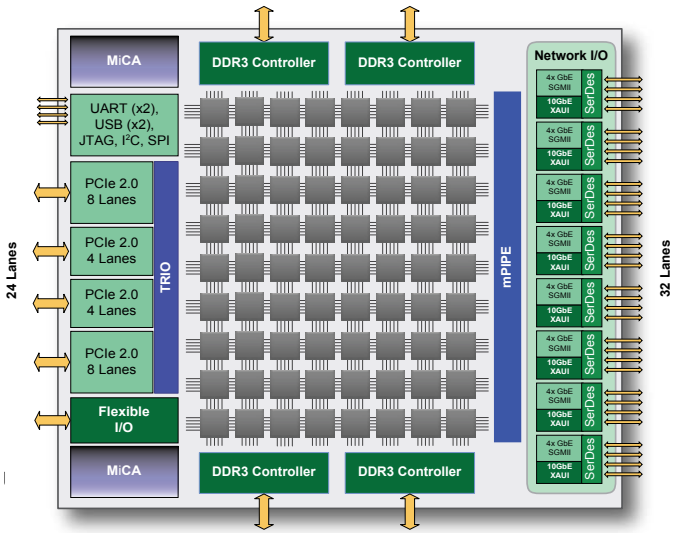
Figure 1.4: Summary of the technology trend.

coordination and processing. Intel's Cloverview [11] (based on dual-core Intel Atom) and Apple's A7 (based on dual-core ARM) are examples of multi-core SoCs. Finally, there are also commercially available SoCs featuring many-core processors. Examples are Sony's PlayStation 3 with IBM Cell processor with 9 cores (2006), Texas Advanced Computing Center's Stampede SoC with 60-core Intel Xeon Phi processor (2013) and SpiNNaker [12] (2013) with more than 100 cores.

As the performance demand kept increasing, single processor SoCs soon became the performance bottleneck, even with up-scaled frequency and multi-core variants. Research took a new direction and thoughts were directed towards integrating multiple full-fledged processors (masters), to manage the application processes, alongside other hardware subsystems. Such platforms are commonly referred to as **multiprocessor systems-on-chip** (MPSoCs). The Lucent's Daytona chip [2], introduced in 2000, is the first known MPSoC, integrating multiple homogeneous processors. Figure 1.3 shows the Daytona MPSoC architecture. Following this breakthrough, there has been a significant drive towards MPSoC development especially in the early half of 2000. Examples included Nexperia [13] from Philips Semiconductor, OMAP [14] from Texas Instruments and Nomadik [15] from STMicroelectronics. Since then, the MPSoC technology has ma-



(a) OMAP multiprocessor system from Texas Instruments (Source: [14]).



(b) Tilera architecture with 72 cores (source: Tilera [18]).

Figure 1.5: Example multiprocessor architectures.

tured to a great extent, growing in complexity and size. Some of the current day MPSoCs are UniPhier [16] from Panasonic Semiconductor, Platform 2012 aka STHORM [17] from STMicroelectronics and Tilera Gx8072 [18] from Tilera Corporation.

Figure 1.4 shows the summary of the four technology trends – transistor technology, microprocessor technology, SoC technology and MPSoC technology. Shown in the same figure is the technology scaling following Dennards' Principles, starting with a gate length of 1mm in 1974 to 5nm in 2014.

The focus of this thesis is on systems-on-chip with multiple processing cores – single processor with multi-/many-cores or multiple processors. The umbrella term **multiprocessor system** is used to represent both these classes of systems. Figure 1.5a shows the *OMAP* multiprocessor system [14] from Texas Instruments, the state-of-the-art multiprocessor system for mobile phones and Figure 1.5b shows a multiprocessor system with

72 cores from Tilera Corporation [18].

Another aspect of multiprocessor systems, relevant to this thesis, is the communication infrastructure, which interconnects the processing elements of these SoCs. Traditionally, multiprocessor systems integrated a shared bus, such as QuickPath Interconnect (QPI) from Intel, STBus from STMicrolelectronics and the AMBA from ARM, to communicate among the processing elements. However, the bottleneck soon shifted from computation capability to large communication delays, severely threatening the integration of additional processing elements. To circumvent this communication problem, a group of researchers from the Stanford University pioneered the implementation of networking concepts for data communication in multiprocessor system in 2002. This concept is known as **network-on-chip** (NoC) [19]. Since then, several commercial NoCs are developed across different research teams. Examples include Æthereal [20] and Hermes [21].

The classification, the design methodologies and the existing design challenges for multiprocessor systems are discussed next.

1.4 Multiprocessor System Classification

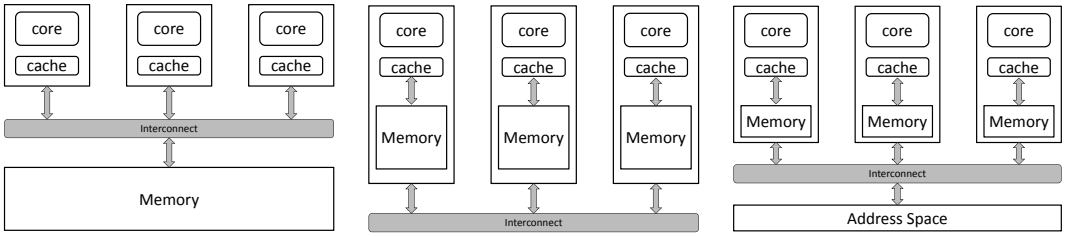
Multiprocessor systems can be classified according to memory distribution, processor type and interconnect network.

1.4.1 Memory-Based Classification

The memory-based classification is shown in Figure 1.6. Multiprocessor systems can be of three types – shared memory, distributed memory and distributed shared memory.

Shared Memory Multiprocessor Systems

In this class of multiprocessor systems, every processing core has its own private L1 and L2 caches, but all the cores share the common main memory. This is shown in Figure 1.6a. All cores in this architecture share a unique memory addressing space, which leads to significant simplification of the programming. Data communication across the cores takes place via the shared memory, accessed using a communication medium, typically a shared bus. Until recently, this was the dominating memory organization for multiprocessor systems with few cores (typically 2 to 4). However, as the gap between



(a) Shared memory. (b) Distributed memory. (c) Distributed shared memory.

Figure 1.6: Memory-based multiprocessor classification.

processor speed and the memory speed increases, and as more interacting parallel tasks are mapped to these cores, the memory bandwidth is becoming a bottleneck.

Distributed Memory Multiprocessor Systems

In this class of multiprocessor systems, every processing core has its own L1 and L2 cache and a private memory space. This is shown in Figure 1.6b. This class of multiprocessor systems allows integration of any number of cores and offers a distinctive advantage over shared memory architecture in terms of scalability. However, programming is complex as compared to the shared memory counterpart. In this architecture, computational tasks mapped on a processing core can operate only on the local data; if remote data is required, the core communicates with other cores. Data communication takes place using a message passing interface through the interconnection network. One of the key characteristics of distributed memory architecture is its non-uniform memory access (NUMA) time. This is dependent on the interconnect network topology, such as Mesh, Tree, Star and Torus.

Distributed Shared Memory Multiprocessor Systems

Distributed shared memory architecture implements the shared memory abstraction on a multiprocessor system. The memories are physically separate and attached to individual core; however, the memory address space is shared across processing cores. This architecture combines the scalability of distributed architecture with the convenience of shared-memory programming. This architecture provides a virtual address space, shared among the cores of the multiprocessor system. This is shown in Figure 1.6c.

Table 1.2: Summary of homogeneous and heterogeneous architectures.

	Homogeneous	**Heterogeneous**
Advantages	Less replication effort, highly scalable	Application specific, high computation efficiency, low power consumption
Limitations	Moderate computation efficiency, high power consumption	Less flexible, less scalable
Compatibility	data parallelism, shared memory architecture, static and dynamic task mapping	task parallelism, message passing interface, static task mapping
Examples	Lucent Daytona [2], Philip Wasabi [22]	Texas Instrument OMAP [14], Samsung Exynos 5 [24]

1.4.2 Processor-Based Classification

Multiprocessor systems can be classified into homogeneous and heterogeneous systems, based on the types of the processing cores.

Homogeneous Multiprocessor Systems

A homogeneous multiprocessor system is a class of SoC where the processing cores are of the same type. Examples of homogeneous multiprocessor systems are Philips Wasabi [22] and Lucent Daytona [2]. One of the advantages of homogeneous multiprocessor systems is the low design replication effort, leading to high scalability. Historically, server multiprocessor systems favored homogeneous processing cores.

Heterogeneous Multiprocessor Systems

Homogeneous architectures are often associated with high area and power requirements [23]; the current trend is, therefore, to integrate heterogeneous processing cores on the same SoC. Such architectures are referred to as heterogeneous multiprocessor systems. This class of systems includes heterogeneity within and across processors. Texas Instrument OMAP [14] is an example of a heterogeneous multiprocessor system that integrates a traditional uni-core processor with a digital signal processor (DSP). Here the heterogeneity is across the different processors. On the other hand, Samsung Exynos 5 [24] is an example of an SoC that integrates a processor with heterogeneous cores (quad-core ARM big.LITTLE architecture [25] i.e., four ARM Cortex A7 cores and four Cortex A15 cores). One of the major limitations of heterogeneous multiprocessor systems is the overhead of integration. Table 1.2 reports the summary of these architectures.

An emerging trend in multiprocessor design is to integrate reconfigurable logic such

as field programmable gate arrays (FPGAs) alongside homogeneous or heterogeneous cores [26]. This class of systems belongs to the heterogeneous category (as the FPGA can be programmed as a processing core, different than the other cores of the system) and are referred to as **reconfigurable multiprocessor systems**. These systems allow FPGA implementation of custom instructions [27] or custom logic [28] specific to the application being executed. Examples include Xilinx Zynq architecture [29].

1.4.3 Network-Based Classification

Finally, multiprocessor systems can be classified based on the topology of the interconnect network forming the communication backbone.

Static Topology-Based Multiprocessor System

In this network architecture, the physical links between the processing cores are determined once at the multiprocessor system design time and remains unaltered throughout. The logical links (i.e. the data flow) can change dynamically based on the data routing strategy. Static network topologies are usually associated with message passing interfaces. Two of the commonly used static topologies for multiprocessor system are

- *Point-to-point:* In a point-to-point network, there exists a direct communication link between every pair of cores of a multiprocessor system. An example is the Intel QuickPath Interconnect.
- *Mesh:* In a mesh network, the processing cores are interconnected through the data routers in a mesh architecture. The data communication takes place through the routers with the communication links shared across multiple processing cores in time or space. Intel Paragon is an example of two dimensional mesh interconnect.

Dynamic Topology-Based Multiprocessor System

In this network architecture, the physical as well as the logical links between the processing cores change dynamically. The dynamic network topologies are usually associated with shared memory interfaces. Two of the commonly used dynamic topologies for multiprocessor systems are

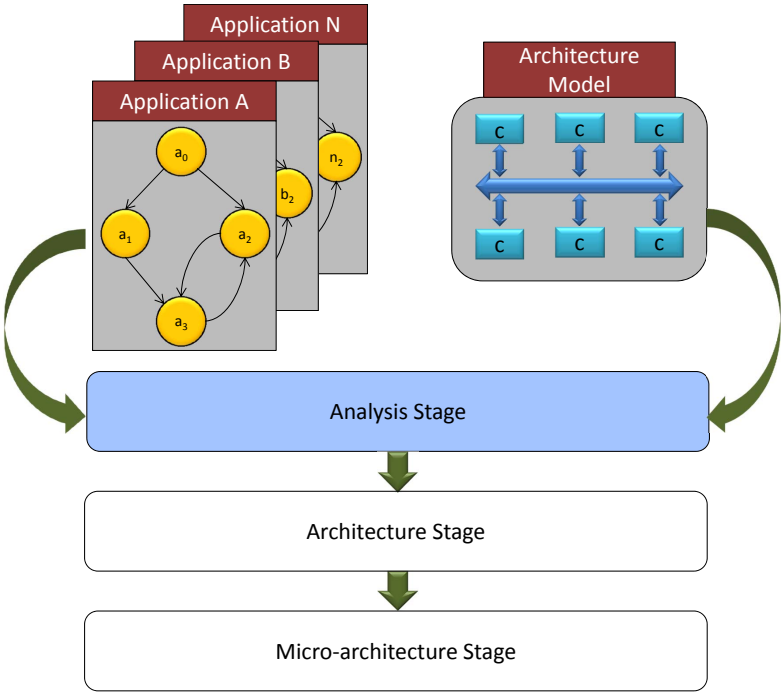


Figure 1.7: Multiprocessor design flow.

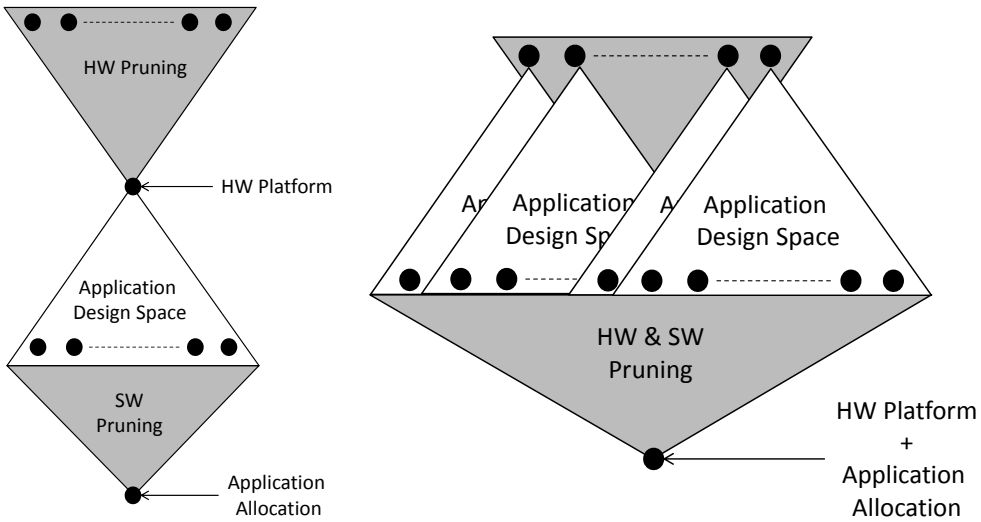
- *Bus:* In bus network, all the processing cores are connected to a shared communication link using a dedicated set of wires. Example is the AMBA bus from ARM.

- *Crossbar:* In a crossbar network, the processing cores are connected to each other in the form of a matrix. The cross points can be programmed dynamically to interconnect different pair of cores of the multiprocessor system.

This thesis focuses on mesh-based multiprocessor systems with distributed memory, using message passing interface for data communication between the processing cores. Both homogeneous and heterogeneous multiprocessor systems (including the reconfigurable category) are studied in this work.

1.5 Multiprocessor System Design Flow

A key concept for a system design is the *orthogonalization of concerns* [30] i.e., the process of separating the different aspects of the design in order to explore the design space of individual aspects more efficiently. Figure 1.7 shows a typical multiprocessor design flow. The first stage of the design flow is the **analysis** stage. The input to this stage are the set of applications supported on the platform and the architectural specification. Applications are usually represented as task graphs with the nodes representing the



(a) Platform-based design. (b) Hardware-software co-design.

Figure 1.8: Multiprocessor system design methodology.

computation tasks of the application and the edges representing the dependency between these tasks as shown in the figure. Application modeling is discussed further in Chapter 2. Architectural specification is provided as a connected graph, with nodes representing the processing cores of the platform and the edges representing the communication medium. There are two aspects of the analysis stage – the hardware aspect and the software aspect. The hardware aspect determines the resources of the platform i.e., the number of processing cores needed for the platform to guarantee performances of the supported applications while satisfying the design budget. The software aspect of the analysis stage determines the allocation i.e., the mapping of the different tasks of each application on the processing cores.

The **architecture** stage determines the architectural details of the resources determined in the analysis stage. For a general purpose processor, the architectural details include determining the instruction set and for the memory organization, the architecture stage addresses the size of the main memory for shared memory architecture or the size of the private memories for the distributed memory architecture. Finally, the **micro-architecture** stage deals with the hardware of the different components. The output of this stage is the register transfer logic (RTL) representation of the multiprocessor system, which can be implemented on an FPGA or an ASIC.

This thesis focuses on the analysis stage of the design flow, which can be further classified into two types – platform-based design and hardware-software co-design.

1.5.1 Platform-Based Design Methodology

Chip design cost has two components – recurring cost and non-recurring cost. The recurring cost is the engineering cost incurred for production of every chip. This cost comprises of the cost for manufacturing, packaging, testing, marketing, distribution and some legal aspects. The **non-recurring engineering** (NRE) cost is the cost incurred once during the production process. This includes the cost for system design and generating the mask set for lithography. Thus, the multiprocessor system design cost is the NRE cost. Since, the cost of mask set comprises the major share of NRE cost and is increasing with every new technology node, it is essential to find a common set of design components that supports not only the specified applications and functions, but also a set of future applications. The design methodology is thus, to determine a hardware platform, driven purely by the NRE cost and shifting the application support objective to software. This design methodology is commonly referred to as **platform-based design** [31, 32] and is shown in Figure 1.8a. The software aspect of the platform-based design is to explore the design space of allocating the available resources of the hardware for every supported application in order to optimize different objectives such as performance, energy and reliability (these objectives are revisited in Section 1.6). Finally the design space is pruned to obtain the optimal allocation strategy for every application.

1.5.2 Hardware-Software Co-Design Methodology

The **hardware-software co-design** methodology [33, 34] performs the hardware pruning and the software pruning steps in an integrated manner to obtain the optimal solution, both with respect to hardware platform and software-based application allocation. This is shown in Figure 1.8b. The NRE design cost is usually considered as a constraint or is integrated with other optimization objectives such as performance, energy and reliability. In its simplest form, the approach starts with a set of hardware design alternatives, each satisfying the NRE economics; the software goal is to explore the design space of allocating the resources for every design alternatives to optimize the required objective. Since different hardware alternatives are explored simultaneously with the software alternatives, the hardware-software co-design is able to achieve better optimization objectives. However, the price paid is the higher exploration time, increasing the time-to-market.

1.6 Design Challenges for Multiprocessor Systems

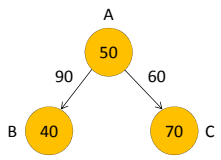
One of the major applications for multiprocessor systems is the embedded systems. Over the last decade, the embedded system market has grown tremendously, both in market share (product valuation) and in market presence (product variety). The embedded market was valued at 121 billion US dollars in 2011 and is expected to reach 194 billion US dollars by 2018. This growth is fueled by increasing demand for embedded systems in emerging sectors such as smart electricity, automobile and medical systems, in addition to the traditional ones such as mobile phones, PDA, laptops and consumer electronics. Energy and reliability are accepted as primary design concerns for embedded devices.

1.6.1 Energy Concern for Multiprocessor Systems

Traditionally, the software aspect of the analysis stage of the design flow has focused on performance optimization. For designs at deep sub-micron technology nodes, such as 65nm and beyond, the focus is gradually shifting from the notion of performance as faster computation [35]. With increasing use of multiprocessor systems in embedded devices, energy optimization has become a primary concern to extend the battery life of these devices. This is forcing system designers to consider energy optimization at every aspect of the design flow. This thesis focuses on the energy optimization at the analysis stage and is orthogonal to the energy optimization techniques at the architecture stage (such as the instruction compression technique of [36]) or at the micro-architecture stage (such as the use of clock gating [37]). The energy minimization at the analysis stage have focused on the following two aspects – reducing the computation energy by slowing down the application on the processing cores using scaled voltage and frequency and reducing the data communication between dependent tasks.

Voltage and Frequency Scaling

The power consumption of a CMOS-based circuit is given by $P = \alpha \cdot F \cdot C \cdot V^2$, where α is the switching factor, F is the frequency, C is the switching capacitance and V is the voltage. Clearly, reducing both the voltage and the frequency by half reduces the total power dissipation by 87.5%. However, the computational time of a job is inversely proportional to the frequency of operation and therefore, reducing the frequency by half increases the



Allocation			Data Communication	Completion Time
A	B	C		
c_0	c_0	c_0	0	160
c_0	c_0	c_1	60	120
c_0	c_1	c_0	90	120
c_0	c_1	c_1	150	160

Figure 1.9: Demonstration of the importance of communication energy.

computation time two fold. The overall effect is that, the energy dissipation, measured as the product of the power dissipation and the computation time, reduces by 75% when both voltage and frequency are scaled by half. This principle of scaling down the energy consumption of a system by reducing the voltage and frequency of operations dynamically during application execution is termed as **voltage and frequency scaling** (VFS). A significant amount of research has been conducted recently to determine the voltage and frequency required for the application execution such that energy consumption is minimized, while satisfying the performance constraint [38–40].

Communication Energy Minimization

When the dependent tasks of an application are mapped to different cores of a multiprocessor system, energy is consumed to transfer data through the interconnect network. This energy is referred to as the communication energy. As established in [41], the communication energy can be as high as upto 40% of the total energy consumption of an application. One way to solve this communication energy problem is to allocate the communicating tasks on the same processing cores. However, this has an impact on the performance. To demonstrate this, a simple example is provided in Figure 1.9. Here, tasks B and C are both ready to be executed after task A completes its execution. The execution time of the tasks (in ms) are labeled as numbers inside the circle. The size of the data (in Kb) required by tasks B and C, from the output of task A are marked in the figure on the arrows from A to B and A to C, respectively. The table in the figure shows the different allocation of the tasks on the two cores (identified as c_0 and c_1) of an example multiprocessor system. The amount of data communicated on the interconnect and the completion time of the application are shown in columns 4 and 5, respectively. Clearly, the allocation of the tasks on a multiprocessor system needs to be performed considering the communication energy and performance trade-off [42].

1.6.2 Reliability Concern for Multiprocessor Systems

Multiprocessor systems are expected to perform error-free operations. In this respect, an **error** of a system is defined as a malfunction i.e., deviation from the expected behavior of the system. Errors are caused by **faults**. However, not every system fault leads to erroneous behavior. This thesis considers malignant faults i.e., the faults that are manifested as errors in the system. Such faults can be classified into three categories – transient, intermittent and permanent. Transient faults are point failures i.e., these faults occur once and then disappear. The faults can occur due to alpha or neutron emissions[1]. Several techniques have been proposed in literature for transient fault-tolerance. Examples are the use of error correction codes for processing cores [43], checkpointing [44], rollback-recovery [45] and duplication with comparison [46]. Intermittent faults are a class of hardware defects, occurring frequently but irregularly over a period of time, due to process, voltage and temperature (PVT) variations. There are techniques that optimize for intermittent faults by analyzing the steady-state availability [47]. Finally, permanent faults, as the name itself indicates, are damages to the circuit caused by such phenomena like electro migration, dielectric breakdowns, broken wires etc. These faults are caused during manufacturing or during the product lifetime due to component wear-out. Hardware redundancy [48] is an effective technique for permanent fault-tolerance.

This thesis focuses on the permanent faults. The permanent defect rates in integrated circuits can be described by the **bathtub curve** as shown in Figure 1.10. Post manufacturing, integrated circuits are characterized by high failure rates. As these circuits are subjected to manufacturing tests (such as stuck-at, at-speed, burn-in, etc., which filters out defective circuits and circuits with short lifetime), the probability of the successful circuits surviving for a longer period of time, increases. The failure rate, therefore, decreases over time. This phase is known as the **infant mortality** period. This is followed by a period of constant failure rate, often referred as **useful life**. The last phase is known as the **wear-out** period and is characterized by increasing fault rate. Recent studies on reliability reveal that, if wear-out is not addressed from early device usage stage (e.g. the beginning of useful life period), circuits can age faster than anticipated with the wear-out phase settling earlier in life (shown by the red dashed line in the figure).

[1]It is to be noted that for FPGA, transient faults in the lookup table content manifest as permanent faults until the content is reprogrammed.

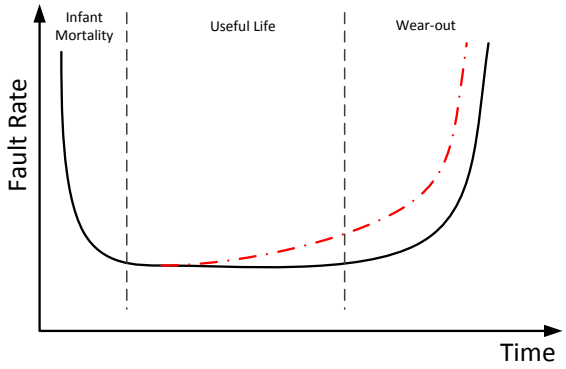


Figure 1.10: Bathtub curve for permanent faults.

Lifetime Reliability of Multiprocessor Systems

At deep sub-micron technology nodes (especially 65nm and lower), sub-threshold leakage and non-ideal voltage and gate oxide scaling are causing the power density to increase with shrinking transistor feature size (contrary to Dennard's principles [4]). This causes localization of the heat (hot spots) and a corresponding increase in the temperature. This increase in chip temperature accelerates wear-out of the semiconductor devices [49–51]. Wear-outs manifest as timing faults and logic faults during the lifetime operation of a device. This is challenging the reliable operation of integrated circuits, reducing their useful life. Timing faults are violations in the design caused due to increase in delay in the circuit due to aging of transistors. A significant research has been conducted recently to analyze and mitigate wear-out related timing faults [47, 52, 53]. This thesis focuses on wear-outs manifested as logic failures and is orthogonal to the techniques for timing fault mitigation. In this context, **lifetime reliability** is defined as the long term reliability of a circuit and is measured in terms of the mean time to permanent failure of the circuit.

As in the case of energy optimization, the reliability optimization can be performed at different levels of the design flow, with the technique adopted at each stage being orthogonal to one another. The micro-architecture techniques involve CMOS device adaptations for wear-out mitigation. Examples include the use of adaptive body biasing technique [54], 22nm tri-gate transistor architecture [55] and device-geometry aware design rule [56]. A summary of the design challenges and wear-out mitigation techniques at the micro-architecture stage is provided in [57]. The architecture-level techniques deal with processor instruction adaptations and scheduling for minimizing different wear-out mechanisms. Examples include intelligent instruction scheduling [58], exploiting instruction timing slacks [59] and the compiler-directed register assignment [60]. Finally, the

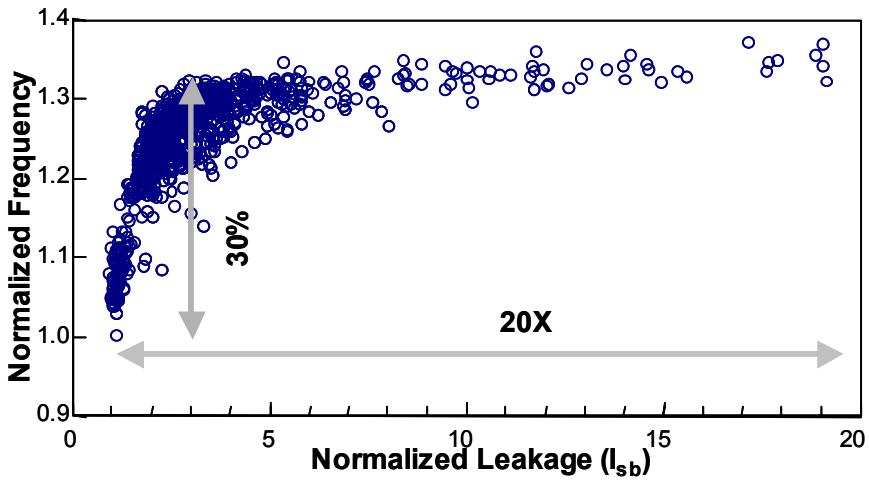


Figure 1.11: Impact of parameter variation on frequency and leakage current [3].

lifetime reliability optimization at the analysis stage involves intelligent application mapping and scheduling [61].

1.6.3 Parameter Variation and its Associated Challenges

Another design challenge at deep sub-micron technology node is concerning parameter variation i.e., the variation of the transistor threshold voltage (V_{th}) and the effective channel length (L_{eff}) [3]. The variation in these parameters effect two key aspects in microprocessor design – highest frequency supported and the leakage power dissipated. The variation in the parameters are attributed to the deviation of the process, voltage and temperature from nominal specification. Process variation is usually caused due to imprecise fabrication process (e.g. lithographic lens aberration) or intra-die irregular dopant density. Voltage variations are caused due to *IR* drop in the voltage supply network. Finally, temperature variations are caused by non-even distribution of load causing some part of the circuit generating more heat than others. Figure 1.11 shows a 30% variation in frequency and a 20x variation in leakage current across the microprocessors fabricated on a 180nm CMOS technology node. These variations are expected to scale up with shrinking feature size.

Several models have been proposed in literature for the parameter variations. Examples include the correlation model of [62] and analytical model of [63]. On of the most significant effect of parameter variation during chip lifetime is timing errors. Recent studies show that with different parts of a chip experiencing different temperature profile, there is a difference in the speed of signal propagation. Specifically, circuits with

high temperature are slower. This causes timing violations in the circuit and forces the device to run at a reduced clock frequency [64]. To accommodate these uncertainties at the system-level design stage, probabilistic variation in worst-case task execution time is usually considered, and the same is integrated within a statistical timing analysis framework to determine the long term period of application execution [65–67].

1.7 Objective of this Thesis

The objective of this thesis is to solve the reliability and the energy concerns for multiprocessor systems – both during the analysis stage of the design flow and also during the in-field operation of the final platform. These two aspects are henceforth referred to as design-time and run-time, respectively.

1.7.1 Design-time Methodology

The design-time methodology of this thesis is highlighted in Figure 1.12. In order to analyze the temperature-related wear-out of different mapping alternatives, a fast and accurate temperature model is developed. This model is characterized using the temperature data obtained from the industry-standard *HotSpot* [68] tool using the floorplan information of a given multiprocessor system. This thermal model is used in the platform-based design methodology along with application and architecture models. Applications are represented as Synchronous Data Flow Graphs (SDFGs) [69] that allow modeling cyclic dependency, multi-input tasks, multi-rate tasks, and pipelined execution. Multiprocessor systems are often designed for multiple applications, many of them enabled concurrently (use-case). Formally, a use-case is defined as a collection of multiple applications that are active simultaneously on a multiprocessor system [70]. To allow optimization for concurrent applications, a set of use-cases is also considered in the design methodology. The multiprocessor platform is represented as a directed graph with nodes representing the processing cores and the edges representing the physical links between the cores. The platform-based design methodology analyses every application and use-case to determine the mapping of the computation tasks on the given platform such that temperature-related wear-outs and energy consumption are jointly minimized. This joint optimization methodology is detailed in Chapter 3.

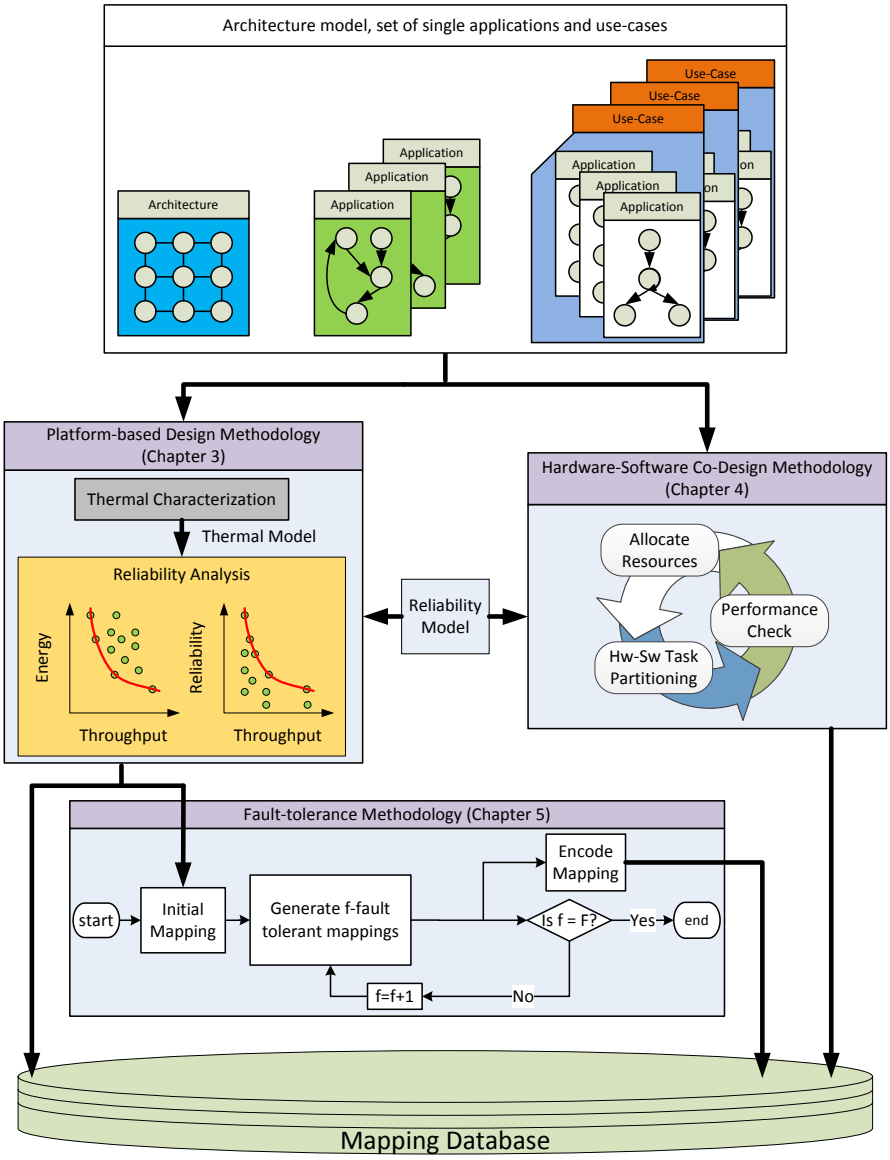


Figure 1.12: Design-time methodology.

As established in Section 1.4.2, an emerging trend in multiprocessor design is reconfigurable multiprocessor system. Application mapping on such platform requires partitioning the computation tasks of the application into software and hardware tasks i.e., deciding on the computation tasks to be executed on the processing cores and those on the reconfigurable area. This hardware-software task partitioning technique for reconfigurable multiprocessor systems with the objective of improving the lifetime reliability is developed in Chapter 4. Based on this approach, a hardware-software co-design methodology is proposed to design reconfigurable multiprocessor systems. A set of applications and use-cases represented as SDFGs is used to determine the minimum number of processing cores and the size of the reconfigurable area required for the system that satisfies

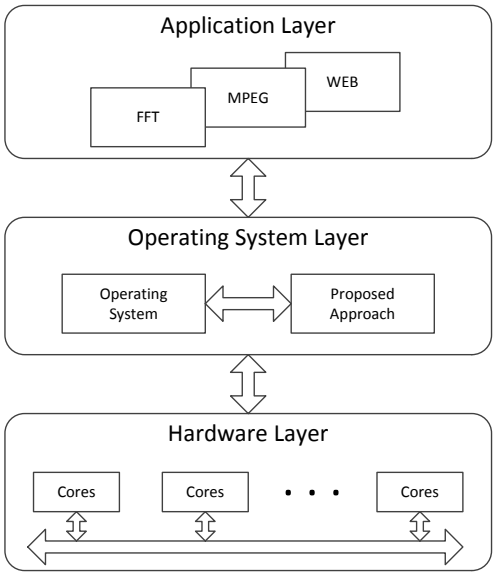


Figure 1.13: Run-time methodology.

the area and energy budget while maximizing the lifetime reliability.

Both the platform-based design and the hardware-software co-design methodologies ameliorate the platform lifetime. These methodologies form part of the **proactive fault-tolerance** approach, which is defined as an approach that prevents or delays the occurrences of fault. A second aspect of the analysis phase of the design methodology is to ensure graceful performance degradation in the presence of faults. This methodology forms part of the **reactive fault-tolerance** approach, which is defined as an approach that deals with resource management post fault occurrences. To achieve this, analysis is performed to determine the minimum energy mapping with the least degradation of performance for every fault scenario. This analysis is performed for every application supported on the system and is detailed in Chapter 5. Finally, to ensure that the performance requirement is satisfied using the pre-analyzed offline mapping at run-time, a self-timed execution-based scheduling technique is proposed for multiprocessor systems.

1.7.2 Run-time Methodology

Figure 1.13 highlights the three layer view of a typical multiprocessor system. At the top, there is the application layer composed of the set of applications supported on the platform. At the bottom is the hardware layer which consists of the processing cores on which the applications are executed. In between these two layers is the operating system or the system software layer, which co-ordinates the application execution on the hardware. The operating system is responsible to assign the computation tasks of an

application on the hardware cores. If the application to be executed on the hardware is already analyzed at design-time, the operating system performs the allocation according to the pre-computed mapping. On the other hand, if a new application (not analyzed beforehand) needs to be executed on the hardware, the operating system needs to determine the correct mapping such that temperature-related wear-out can be minimized. To achieve this, a machine learning-based run-time approach is developed in Chapter 6. The run-time approach is built as a part of the operating system and interacts with the application layer on one hand to determine the performance requirement and with the hardware on the other hand to collect the thermal data. Based on this information, the machine learning agent adjusts the voltage-frequency control and the thread allocation of the operating system to effectively reduce thermal emergencies.

1.8 Key Contributions of This Thesis

Following are the contributions that have led to this thesis:

1. *A fast and accurate thermal model* to estimate the temperature of the cores of a multiprocessor system from a given floorplan. The model incorporates both the temporal and the spatial dependencies. The predicted temperature from this model is shown to be within 4% of the state-of-the-art accurate thermal model.[2] This thermal model is published in [71].

2. *A gradient-based fast heuristic* to jointly optimize lifetime reliability and energy computation of a given multiprocessor system with applications represented as SDFGs. The proposed methodology distributes the cores of the system among concurrently executing applications using integer linear programming. The gradient-based approach increases lifetime by an average 47% and minimizes energy consumption by 24% with respect to the existing techniques. This work is published in [72].

3. *A hardware-software task partitioning framework* for reconfigurable multiprocessor system to decide on the computation tasks to be executed on the processing cores and those on the reconfigurable area. The objective is to improve the lifetime reliability of the platform while exploring the trade-off between lifetime reliability and

[2]The state-of-the-art thermal models are too computation intensive to be included in the design space exploration process.

the reliability considering transient faults with checkpointing-based fault-tolerance. The approach is shown to improve the lifetime reliability by an average 60% with less than 15% sacrifice of the reliability considering transient faults. This work is published in [73].

4. *A hardware-software co-design methodology* to determine the minimum number of cores and the size of the reconfigurable area for a reconfigurable multiprocessor system such that, the lifetime reliability is maximized while satisfying the area and energy budget. This methodology is shown to improve lifetime reliability by an average 65% for single applications and an average 70% for use-cases with 25% fewer cores and 20% less reconfigurable area as compared to the existing hardware-software co-design approaches.

5. *A design-time (offline) multi-criteria optimization technique* for application mapping on multiprocessor systems to minimize the energy consumption for all processor fault-scenarios while providing graceful throughput degradation. The proposed technique is shown to minimize energy consumption by an average 22% as compared to the existing techniques. Parts of this contribution are first published in [74] and [75], and later improved and published in [76] and [77], respectively.

6. *A scheduling technique based on self-timed execution* to minimize the schedule storage and construction overhead at run-time. The scheduling technique is shown to minimize schedule construction time by 95% and storage overhead by 92%. This work is published in [77].

7. *A reinforcement learning-based Inter-and intra-application thermal management* to control the peak temperature as well as thermal cycling using thread-to-core allocation (through CPU affinity) and dynamic frequency control (through CPU governors). Results demonstrate that the proposed approach minimizes average temperature, peak temperature and thermal cycling, improving the mean time to failure by an average of 2x for intra-application and 3x for inter-application scenarios when compared to existing thermal management techniques. Furthermore, the dynamic and static energy consumption are also reduced by an average 10% and 11%, respectively. This work is published in [78].

The remainder of this thesis is organized as follows. Chapter 2 provides a detailed literature survey of the existing techniques for reliability and energy optimization in multiprocessor systems, identifying the gaps in these studies. Chapter 3 introduces the reliability and energy-aware platform-based design methodology for multiprocessor system. The hardware-software co-design methodology for reconfigurable multiprocessor system is introduced in Chapter 4. The reactive fault-tolerance methodology is discussed next in Chapter 5. The run-time thermal management is detailed in Chapter 6. Finally, Chapter 7 concludes this thesis with an overview of the future research directions.

CHAPTER 2

Literature Survey on System-level Fault-tolerant Techniques

Multiprocessor systems are becoming increasingly more complex. As mentioned in Chapter 1, lifetime reliability is a growing concern in these systems as escalating power density, and hence temperature, continues to accelerate wear-out leading to a growing prominence of permanent device defects. This problem needs to be addressed both for applications known apriori at the platform design-time and for new applications enabled by users at run-time. The key to success of the analysis stage of the design-flow is a good model of an application that allows accurately predicting its performance on a given hardware, without actually going through the architecture and the micro-architecture stages for synthesizing the system. Several abstract models have been proposed to represent applications. Refer to [79] for a comparative study of the different data flow models based on their expressiveness and succinctness, efficiency of implementation, and analyzability. Additionally, reliability analysis and management require a reliability model that is able to predict the system lifetime from the temperature obtained while executing an application. This system-level reliability model needs to incorporate the transistor-level wear-out phenomena and their dependency on temperature.

The remainder of this chapter is organized as follows. Section 2.1 gives an introduction to synchronous data flow graphs (SDFGs) that is used in this thesis as application models for design-time analysis. The system-level reliability modeling is discussed next

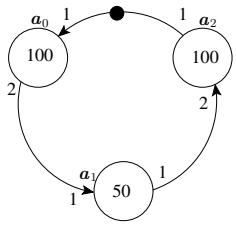


Figure 2.1: Application SDFG.

in Section 2.2. This is followed by detailed discussions on the existing design-time based and run-time based reliability-energy joint optimization techniques in Sections 2.3 and 2.4, respectively, highlighting the limitations that are addressed in this thesis. Finally, the chapter is concluded in Section 2.5.

2.1 Introduction to Synchronous Data Flow Graphs

Synchronous Data Flow Graphs (SDFGs, see [69]) are often used for modeling modern DSP applications [80] and for designing concurrent multimedia applications implemented on multiprocessor systems. Both pipelined streaming and cyclic dependencies between tasks can be easily modeled in SDFGs. SDFGs allow analysis of a system in terms of throughput and other performance properties e.g., latency and buffer requirements [81].

The nodes of an SDFG are called **actors**; they represent functions that are computed by reading **tokens** (data items) from their input ports and writing the results of the computation as tokens on the output ports. The number of tokens produced or consumed in one execution of an actor is called port **rate**, and remains constant. The rates are visualized as port annotations. Actor execution is also called **firing**, and requires a fixed amount of time, denoted with a number in the actors. The edges in the graph, called **channels**, represent dependencies among different actors.

Figure 2.1 shows an example of a SDFG. There are three actors in this graph. In the example, $\boldsymbol{a}_0$ has an input rate of 1 and output rate of 2. An actor is called **ready** when it has sufficient input tokens on all its input edges and sufficient buffer space on all its output channels; an actor can only fire when it is ready. The edges may also contain **initial tokens**, indicated by bullets on the edges, as seen on the edge from actor $\boldsymbol{a}_2$ to $\boldsymbol{a}_0$ in Figure 2.1. Formally, an SDFG is a directed graph $\mathcal{G}_{app} = (A, C)$ consisting of a finite set A of actors and a finite set C of channels.

One of the most interesting properties of SDFGs relevant to this thesis is throughput.

Throughput is defined as the inverse of the long term period i.e., the average time needed for one iteration of the application. (An iteration is defined as the minimum non-zero execution such that the original state of the graph is obtained.) This is the performance parameter used in this thesis. The following properties of an SDFG are defined.

Definition 1 (Repetition Vector) *Repetition Vector* RV *of an SDFG,* $\mathcal{G}_{app} = (A, C)$ *is defined as the vector specifying the number of times actors in* A *are executed for one iteration of SDFG* G_{app}*. For example, in Figure 2.1,* $RV[\mathbf{a}_0\ \mathbf{a}_1\ \mathbf{a}_2] = [1\ 2\ 1]$.

Definition 2 (Application Period) *Application Period* Per(A) *is defined as the time SDFG,* $\mathcal{G}_{app} = (A, C)$ *takes to complete one iteration on an average.*

The period of an SDFG can be computed by analyzing the maximum cycle mean (MCM) of an equivalent homogeneous SDFG (HSDFG). The period thus computed gives the minimum period possible with infinite hardware resources e.g. buffer space. If worst-case execution time estimates of each actor are used, the performance at run-time is guaranteed to meet the analyzed throughput. Self-timed strategy is widely used for scheduling SDFGs on multiprocessor systems. In this technique, the exact firing of an actor on a core is determined at design-time using worst-case actor execution-time. The timing information is then discarded retaining the assignment and ordering of the actors on each core. At run-time, actors are fired in the same order as determined from design-time. Thus, unlike fully-static schedules, a self-timed schedule is robust in capturing the dynamism in actor execution time. The self-timed execution of an SDFG consists of a transient phase followed by a periodic or steady-state phase [82].

2.2 Wear-out Related Failure Modeling

This section reviews the failure modeling of multiprocessor systems, which will form the basis for comparing the existing works on reliability optimization. The multiprocessor system-level reliability modeling is dependent on the core-level reliability modeling, which in-turn is dependent on the device-level reliability modeling. These are discussed next.

2.2.1 Device-Level Reliability Modeling

Following are the dominant wear-out related failure mechanisms.

Electromigration

Electromigration refers to the movement of metal atoms from the interconnect wires and vias due to the flow of current, temperature gradient and electric diffusion, causing open and short circuits in the interconnect. The mean time to failure (MTTF) due to electromigration is given by the following equation [83, 84].

$$MTTF_{EM} = \frac{A_{EM}}{J^n} \exp\left(\frac{E_{a_{EM}}}{KT}\right) \tag{2.1}$$

where A_{EM} is a material-dependent constant, J is the current density, n is empirically determined constant with a typical value of 2 for stress related failures, $E_{a_{EM}}$ is the activation energy of electromigration, K is the Boltzman's constant and T is the temperature. The current density J is determined as

$$J = \frac{f \cdot C \cdot V_{dd} \cdot P_t}{W \cdot H} \tag{2.2}$$

where C is the parasitic capacitance, V_{dd} is the supply voltage, f is the clock frequency, P_t is the probability of line toggling in a clock cycle, W is the width of the metal line and H is the thickness of the metal line.

Negative Bias Temperature Instability

Negative bias temperature instability refers to the shift in the threshold voltage and saturation current in p-channel MOS (PMOS) devices after an extended period of stress caused by the application of negative voltage across the gate to channel region [85, 86]. The MTTF due to negative bias temperature instability is given by the following equation [87]

$$MTTF_{NBTI} = \frac{A_{NBTI}}{(V_{GS})^{\gamma}} \exp\left(\frac{E_{a_{NBTI}}}{KT}\right) \tag{2.3}$$

where A_{NBTI} is a constant dependent on the fabrication process, V_{GS} is the gate voltage, γ is the voltage acceleration factor and $E_{a_{NBTI}}$ is the activation energy of negative bias temperature instability.

Hot Carrier Injection

There are three types of carrier injection – channel hot electron injection, drain avalanche hot carrier injection and secondary generated hot electron injection. Channel hot electron injection refers to the escape of electrons from the channel causing a degradation of the Si-SiO_2 interface [88, 89]. Drain avalanche hot carrier injection refers to the gate

oxide degradation due to the electron and hole gate currents caused due to the impact ionization [88]. Finally, secondary generated hot electron injection refers to the injection of minority carriers due to secondary impact ionization [88]. The MTTF due to hot carrier injection is given by

$$MTTF_{HCI} = A_{HCI} \exp\left(\frac{\theta}{V_{DS}}\right) \tag{2.4}$$

where A_{HCI} and θ are empirically determined constants and V_{DS} is the drain to source voltage.

Time Dependent Dielectric Breakdown

Time dependent dielectric breakdown refers to the degradation of the SiO_2 insulating layer between the gate and the conducting channel of the MOS device. The applied voltage and the tunneling electrons create defects in the volume of the oxide film, which accumulates over time triggering a sudden loss of dielectric properties. The exact physical mechanism behind the degradation is, however, still an open question. The MTTF due to time dependent dielectric breakdown is given by [84]

$$MTTF_{TDDB} = A_{TDDB} \cdot A_G \cdot \left(\frac{1}{V_{GS}}\right)^{\alpha-\beta T} \exp\left(\frac{X}{T} + \frac{Y}{T^2}\right) \tag{2.5}$$

where $\alpha,\ \beta,\ X$ and Y are the fitting parameters, A_G is the surface area of the gate oxide and A_{TDDB} is an empirically determined constant.

Stress Migration

Stress migration refers to the motion of atoms in metal wires due to mechanical stress caused by the mismatch of temperature between the metal and the dielectric material. The MTTF due to stress migration is given by

$$MTTF_{SM} = A_{SM}|T_0 - T|^{-n} \exp\left(\frac{E_{a_{SM}}}{KT}\right) \tag{2.6}$$

where A_{SM} is a material dependent constant, T_0 is the metal deposition temperature and $E_{a_{SM}}$ is the activation energy of stress migration.

Thermal Cycling

Thermal cycling refers to the wear-out caused by thermal stress due to a mismatched coefficient of thermal expansion of the adjacent material layers. Thermal cycling related MTTF is computed in three steps.

1. Calculating the thermal cycles from a thermal profile using Downing simple rainflow counting algorithm [90].

2. Calculating, from each thermal cycle, the number of cycles to failure using Coffin-Manson's rule [91].

$$N_{TC}(i) = A_{TC}\left(\delta T_i - T_{Th}\right)^{-b} e^{\frac{E_{a_{TC}}}{KT_{max}(i)}} \tag{2.7}$$

   where $N_{TC}(i)$ is the number of cycles to failure due to i^{th} thermal cycle, A_{TC} is an empirically determined constant, δT_i is the amplitude of the ith thermal cycle, T_{Th} is the temperature at which elastic deformation begins, b is the Coffin-Manson exponent constant, $E_{a_{TC}}$ is the activation energy of thermal cycling and $T_{max}(i)$ is the maximum temperature in the ith thermal cycle.

3. Calculating the MTTF using Miner's rule [92].

$$MTTF = \frac{N_{TC}\sum_{i=1}^{m} t_i}{m} \tag{2.8}$$

   where t_i is the time for the ith thermal cycle, m is the number of thermal cycles obtained in step 1 and N_{TC} is the effective cycles to failure determined using

$$N_{TC} = \frac{m}{\sum_{i=1}^{m} \frac{1}{N_{TC}(i)}} \tag{2.9}$$

   Combining Equations 2.7-2.9,

$$MTTF = \frac{A_{TC}\sum_{i=1}^{m} t_i}{Thermal\ Stress} \tag{2.10}$$

   where *Thermal Stress* is an indication of the stress experienced due to the thermal cycling. This is obtained using the following equation.

$$Thermal\ Stress = \sum_{i=1}^{m} \left(\delta T_i - T_{Th}\right)^b \times e^{\frac{-E_a}{KT_{max}(i)}} \tag{2.11}$$

2.2.2 Core-Level Reliability Modeling

Core-level reliability modeling approach is to combine the device level reliability models to estimate the mean time to failure of the cores of a multiprocessor system.

The fault density at the device level is typically characterized by Weibull or Lognormal distribution. For example, time dependent dielectric breakdown follows Weibull distribution and electromigration follows Lognormal distribution. The distributions for other wear-out mechanisms are not known with certainty. The reliability computation is demonstrated for these two types of distribution.

Weibull Distribution

The reliability of a core considering Weibull distribution for the fault density is given by

$$R(t) = e^{-\left(\frac{t}{\eta}\right)^{\beta}} \tag{2.12}$$

where η is the scale parameter and β is the shape parameter. When temperature is not a constant, but varies over time, any time interval 0 to t can be split into n disjoint time intervals $\{[0, t_1), [t_1, t_2). \cdots [t_{n-1}, t_n)\}$ such that the temperature is constant within each interval. Let T_i be the temperature at the time interval $[t_i, t_{i+1})$. The scale parameter for the Weibull distribution for each interval is given by

$$\eta_i = \frac{MTTF_{WO}(T_i)}{\Gamma(1+\beta)} \tag{2.13}$$

where Γ is the gamma function and $MTTF_{WO}$ is the MTTF with specific wear-out (WO) type under consideration, i.e.

$$WO = \begin{cases} EM & \text{for electromigration} \\ NBTI & \text{for negative bias temperature instability} \\ \cdots \end{cases} \tag{2.14}$$

The reliability is therefore given by [93]

$$R(t) = e^{-\left(\sum_{i=1}^{n} \frac{t_i - t_{i-1}}{\eta_i}\right)^{\beta}} \tag{2.15}$$

Impact of process variation: The expression for reliability is derived assuming a constant value for the shape parameter β. When process variation is considered, β is not constant, but is given by a Gaussian distribution function $\phi(\mu_g, \sigma_g)$, where μ_g is the mean and σ_g^2 is the variance. The reliability of a component (core) with N devices considering process variation is given by [94]

$$R(t) = e^{-N\left(\sum_{i=1}^{n} \frac{t_i - t_{i-1}}{\eta_i}\right)^{\mu_g - \frac{\sigma_g^2}{2} ln\left(\sum_{i=1}^{n} \frac{t_i - t_{i-1}}{\eta_i}\right)}} \tag{2.16}$$

Lognormal Distribution

The reliability of a core considering Lognormal distribution for the fault density is given by

$$R(t) = \frac{1}{2} - \frac{1}{2} erf\left(\frac{ln(t) - \mu}{\sqrt{2\sigma^2}}\right) \tag{2.17}$$

where μ is the scale parameter, σ is the shape parameter and erf is the error function. Considering a time distribution of temperature as before, the reliability is

$$R(t) = \frac{1}{2} - \frac{1}{2} erf \left(\frac{ln(t) + ln\left(\frac{\sum_{i=1}^{n} \frac{t_i - t_{i-1}}{e^{\mu_i}}}{t} \right)}{\sqrt{2\sigma^2}} \right) \tag{2.18}$$

where the scale parameter μ_i for the ith interval is

$$\mu_i = ln\left(MTTF_{WO}(T_i)\right) - \frac{\sigma^2}{2} \tag{2.19}$$

The reliability of a core with N devices using Lognormal distribution for fault density is given by [93]

$$R(t) = e^{N \int_{-\infty}^{\infty} f(x) ln\left(\frac{1}{2} - \frac{1}{2} erf\left(\frac{ln(t)-\mu}{\sqrt{2x^2}}\right)\right) dx} \tag{2.20}$$

where $f(x)$ is the probability density function of μ. This equation is too complex to integrate in the reliability computation for systems. Furthermore, recent studies reveal that for large number of devices per component, Weibull distribution provides more accurate modeling than Lognormal distribution and therefore has been adopted for most of the existing works on reliability optimization.

MTTF of a Core

The mean time to failure for a core $\boldsymbol{c}_i$ is therefore

$$MTTF_i = \int_0^{\infty} R(t) dt \tag{2.21}$$

2.2.3 System-Level Reliability Modeling

The system-level reliability modeling is to combine the reliability of the different cores to determine the mean time to failure of the multiprocessor system. Some of the most widely used MTTF computation techniques are highlighted here.

Max-Min Approach: One of the most widely adopted approaches for MTTF analysis is the *Max-Min* approach [95–97], where the MTTF of a system is approximated to the minimum MTTFs of the different cores i.e.,

$$MTTF_{sys} = \min_i MTTF_i \tag{2.22}$$

Additive Refinement Approach: In the iterative approach, the mean time to failure is determined iteratively, considering one failure at a time [98]. After every failure, the tasks on the faulty core are remapped to the other active cores. This changes the operating temperature and hence the shape parameter of the fault distribution functions. The new

Algorithm 1 Iterative reliability computation

Input: Application and architecture graphs
Output: MTTF of the multiprocessor systems
1: Initialize $MTTF_{sys} = 0$
2: Map and schedule the application on the system
3: **while** performance requirement is satisfied **do**
4: **for all** core c_i of the system **do**
5: Determine $MTTF_i$
6: **end for**
7: $MTTF_{sys} = MTTF_{sys} + \min\{MTTF_i\}$
8: Task migration and determine new schedule
9: **end while**

MTTF are computed for the cores and the minimum MTTF of all the cores is added to system MTTF. This process is repeated until the performance of an application drops below an acceptable limit. Algorithm 1 provides the pseudo-code of the approach.

Multi-Convolution Integral Approach: The reliability of a multiprocessor system with l failures is given by [99]

$$R^{sys}_{N_c-l}(t) = \int_0^t dt_1 \int_{t_1}^t dt_2 \cdots \int_{t_{l-1}}^t R^{sys}_{N_c-l}(t, \mathbf{t}_l) dt_l \tag{2.23}$$

where N_c is the number of cores of the multiprocessor system, l is the number of failures, $\mathbf{t}_l = (t_1, t_2, \cdots, t_l)$, where t_i is the occurrence of the ith failure and

$$R^{sys}_{N_c-l}(t, \mathbf{t}_l) = R^{sys}_{N_c-l}(t|\mathbf{t}_l) \cdot g^{sys}_{N_c}(t_1) \cdot g^{sys}_{N_c-1}(t_2|t_1) \cdots g^{sys}_{N_c-l+1}(t_l|t_1, t_2, \cdots t_{l-1}) \tag{2.24}$$

where $R^{sys}_{N_c-l}(t|\mathbf{t}_l)$ is the probability that a core survives at time t given the system experiences l failures and $g^{sys}_{N_c-r+1}(t_r|t_1, t_2, \cdots t_{r-1})$ is the probability that a system containing $N_c - r + 1$ working cores experiences the rth failure at time t_r with the past $r-1$ failures occurring at $t_1, t_2, \cdots t_{r-1}$. The MTTF of the system is

$$MTTF_{sys} = \int_0^\infty \sum_{l=N_c^{min}}^{N_c} R^{sys}_{N_c-l}(t) dt \tag{2.25}$$

where N_c^{min} is the minimum number of cores required to satisfy the performance of a system. For a gracefully degrading system, $N_c^{min} = 1$.

Monte-Carlo Simulation Approach: The MTTF of a system can be derived using Monte-Carlo simulation using survival lattice to describe system structure. The time to failure of a component assuming Weibull distribution is given by [93, 100]

$$TTF = \frac{t}{\sum_{i=1}^{n} \frac{t_i - t_{i-1}}{\eta_i}} e^{\frac{\mu - \sqrt{\mu^2 + 2\sigma ln\left(-\frac{ln(1-u)}{N_c}\right)}}{\sigma^2}} \tag{2.26}$$

where u is a uniform random number in $[0, 1]$ representing the expected lifetime during system-level simulation. The system-level MTTF is determined using the survival lattice.

The system reliability can be calculated as the percentage of trials for which system survives longer than TTF.

2.3 Design-time Based Reliability and Energy Optimization

As established in Chapter 1, the design-time methodology addresses three aspects – reliability and energy-aware platform-based design, reliability and energy-aware hardware-software co-design and energy-aware mapping for proactive fault-tolerance. The existing studies on these three aspects are discussed next.

2.3.1 Existing Approaches on Platform-Based Design

Since temperature has a significant impact on device wear-out, there are quite some research studies on offline task allocation techniques for temperature minimization. A thermal-aware task mapping technique is proposed in [101] based on the *HotSpot* tool. A mixed integer linear programming (MILP)-based task mapping and scheduling is proposed in [96] that solves the steady-state and spatial temperature dependency from the resistive capacitive (RC) thermal equivalent model. A fast event-driven approach is proposed in [102] to estimate the temperature of multiprocessor systems using prebuilt look-up tables and predefined leakage calibration parameters. All these techniques determine the steady-state temperature only. A temperature-aware technique is proposed in [103] to distribute the idle time in order to control the power consumption and hence the temperature. Both the transient and the steady-state phases are modeled in these approaches. However, spatial dependency is not considered.

Since different wear-out mechanisms are influenced by temperature differently, there are studies that optimize lifetime reliability, directly considering these wear-out mechanisms through intelligent task mapping. A reliability estimation technique is proposed in [98] for application specific multiprocessor systems. The model considers multiple failures by incorporating the change in fault density with core failures. A slack allocation technique is proposed in [104] to improve the lifetime reliability of NoC-based multiprocessor systems. The proposed technique exploits the critical quantity slack arising from execution and storage resources to increase the MTTF. A simulated annealing based technique is proposed in [95] to maximize the lifetime reliability of a multipro-

cessor system. The steady-state temperature values are determined using the *HotSpot* tool for all combinations of the active tasks on different processors. These temperature data are stored in a lookup and used during the optimization step. Ant-colony based optimization technique is proposed in [105] to determine the task mapping that maximizes lifetime defined as the time to the first failure. This technique has shown that the lifetime of a multiprocessor system using temperature-aware optimization technique can be significantly lower than when lifetime is explicitly optimized. A simulated annealing based energy-reliability joint optimization technique is proposed in [106]. based on the temperature model of [95].

The technique in [97] uses eigen value decomposition based approach to determine the steady-state dynamic temperature profile using a time-varying and periodic power profile. The approach is shown to be the most accurate among all the existing techniques. Based on this temperature model, a simple heuristic is proposed to maximize the lifetime of a multiprocessor system considering thermal cycling-related wear-out effects. Finally, there are also techniques to optimize the lifetime of multiprocessor systems considering transient and intermittent faults. A resource management technique is proposed in [107] to minimize processor wear-out, simultaneously providing tolerance for transient and intermittent faults. A genetic algorithm based lifetime optimization technique is proposed in [108]. The approach determines the voltage and frequency of the cores of a multiprocessor system to maximize the lifetime and minimize the soft-error susceptibility. Markov-decision based multiprocessor steady-state availability is derived in [47] considering intermittent faults. Table 2.1 summarizes these related works and highlights the objectives of this thesis.

2.3.2 Existing Approaches on Hardware-Software Co-Design

A hardware-software co-synthesis of fault-tolerant systems is proposed in [109]. The proposed approach uses task duplication with comparison and re-execution as the fault-tolerant mechanism. A design methodology is proposed in [110] to handle conditional execution in multi-rate embedded systems and selectively duplicates critical tasks to correct transient errors. A technique is proposed in [111] to determine the cost, performance and reliability trade-offs for multiprocessor system considering permanent faults. A design space exploration of multimedia multiprocessor systems is proposed in [112]. The

Table 2.1: Related works on reliability and energy-aware platform-based design.

Related Works	**Temperature Model**	**Optimization Objective**	**Reliability Modeling**	**Application Model**	**Architecture Model**
Gu et al. [98]	steady-state & spatial	lifetime reliability	*Additive Refinement*	independent DAGs	static heterogeneous
Meyer et al. [104], Huang et al. [95], Hartman et al. [105]	steady-state & spatial	lifetime reliability	*Max-Min*	independent DAGs	static homogeneous
Huang et al. [106]	steady-state & spatial	lifetime reliability & energy	*Max-Min*	independent DAGs	static homogeneous
Ukhov et al. [97]	transient, steady-state, temporal & spatial	lifetime reliability & energy	*Max-Min*	independent DAGs	static homogeneous
Chou et al. [107], Das et al. [47, 108]	steady-state & spatial	lifetime reliability, transient & intermittent faults	*Max-Min*	independent DAGs	static homogeneous
Objectives of this thesis [71, 72]	transient, steady-state, temporal & spatial	lifetime reliability & energy	*Additive Refinement*	independent & concurrent SDFGs	static homogeneous

proposed approach explores the trade-off between different metrics such as performance, energy and cost while incorporating soft-error tolerance in the optimization process.

A system level reliability analysis technique is proposed in [113] considering process re-execution in software and selective hardening of hardware for fault-tolerance. Based on this, a design optimization heuristic is proposed to select the fault-tolerant architecture and the task mapping such that, the overall cost is minimized. A system-level synthesis flow is proposed in [114] for the design of reliable embedded systems. The methodology explores different hardening strategies under a given user level reliability specification. The strategy with the least resource utilization is selected and the given application is mapped on the resulting platform to optimize reliability. All these techniques determine the multiprocessor platform to maximize the fault-tolerance capability considering transient and permanent faults.

There are very few research studies on the hardware-software co-design methodology for proactive fault-tolerance considering temperature-related wear-out. A system-level design methodology is proposed in [115] for the automatic synthesis of reliable embedded systems. The proposed methodology addresses the following: selection of resources with different reliability, area and latency parameters; and mapping of a data flow application on the platform to simultaneously optimize reliability, area and latency using multi-objective evolutionary algorithm. A co-design methodology is proposed in [116] to determine the minimum resources required to improve system lifetime measured as MTTF. Table 2.2 summarizes the related works and the objectives of this thesis.

Table 2.2: Related works on reliability and energy-aware hardware-software co-design.

Related Works	**Optimization Objective**	**Reliability Modeling**	**Fault-tolerance**	**Multiprocessor Platform**	**Application Model**
Dave et al. [109], Xie et al. [110], Bolchini et al. [114]	area	–	reactive	static heterogeneous	independent DAGs
Bolchini et al. [111], Stralen et al. [112], Izosimov et al. [113]	reliability	–	reactive	static homogeneous	independent DAGs
Glaß et al. [115]	lifetime reliability	*Convolution Integral*	proactive	static homogeneous	independent DAGs
Zhu et al. [116]	area & lifetime reliability	*Additive Refinement*	proactive	static heterogeneous	independent DAGs
Objectives of this thesis [73]	lifetime reliability	*Max-Min*	proactive & reactive	homogeneous & reconfigurable	independent & concurrent SDFGs

2.3.3 Existing Approaches on Reactive Fault-tolerance

The offline reactive fault-tolerant techniques address the task allocation problem on a multiprocessor system considering the occurrences of permanent faults. A multi-objective optimization approach is proposed in [117] to jointly optimize cost, time and dependability. The application task graph is extended with certain nodes of the graph replicated to allow fault tolerance. The dependability of the replicated task graph is evaluated considering resource crash as the fault model. The approach performs design space exploration using the replicated graph in a genetic algorithm framework to maximize the dependability, while satisfying the design cost and execution time constraint. A fixed order Band and Band reconfiguration technique is studied in [118]. Cores of the target architecture are partitioned into two bands. When one or more cores fail, tasks on these core(s) are migrated to other functional core(s) determined by the band in which these tasks belong. The core partitioning strategy is fixed at design-time and is independent of the application throughput requirement. A re-execution slot based reconfiguration mechanism is studied in [119]. Normal and re-execution slots of a task are scheduled at design-time using evolutionary algorithm to minimize certain parameters like throughput degradation. At run-time, tasks on a faulty core migrate to their re-execution slot on a different core. However, a limitation of this technique is that the schedule length can become unbounded for high fault-tolerant systems. Task remapping technique based on offline computation and virtual mapping is proposed in [120]. Here, task mapping is performed in two steps – determining the highest throughput mapping followed by the generation of a virtual mapping to minimize the cost of task migration to achieve this highest throughput mapping. An ILP based approach is presented in [121]. Energy opti-

Table 2.3: Related works on design-time reactive fault-tolerant techniques.

Related Works	Fault-tolerant Mapping & Scheduling	Energy Optimization	Migration Overhead	Application Model	Throughput Degradation
Jhumka et al. [117]	task mapping only	×	×	DAGs	×
Yang et al. [118]	task mapping only	×	√	DAGs	×
Huang et al. [119]	mapping & scheduling	×	×	DAGs	×
Lee et al. [120]	task mapping only	×	×	DAGs	√
Wei et al. [121]	task mapping only	√	×	DAGs	×
Objectives of this thesis ([74–77])	mapping & scheduling	√	√	SDFGs	√

mization is performed under execution time constraint which incorporates fault-tolerance overhead using check-pointing based recovery model. Table 2.3 summarizes the existing reactive fault-tolerant techniques and the objectives of this thesis.

2.4 Run-time Based Reliability and Energy Optimization

A dynamic reliability management technique is proposed in [122], where workload characteristics and thermal information are used to project the degradation caused by various failure mechanisms. The relationship between temperature and voltage and frequency of operation is formulated in [123]. Based on this, an online heuristic is proposed to determine the voltage and frequency of the cores to minimize the temperature. In [124], a distributed dynamic thermal management technique is proposed to avoid thermal hot spots that accelerate thermal aging and transient faults. The proposed technique consists of multiple agents, each managing a cluster of the many-core architecture.

A run-time task mapping technique is proposed in [125] to minimize the wear-out by utilizing the wear-out sensors. The proposed technique computes task mapping at regular intervals and when a component fails. The scheduling decision is left to the operating system. An online approach is proposed in [126] to minimize wear-out considering different aging mechanisms. The power of a core is used to estimate its temperature and a simple heuristic is proposed to dynamically manage peak temperature and thermal cycling. Both these techniques are based on simulation of a real multiprocessor system.

A proactive thermal management policy is proposed in [127] to balance the temperature on the die. The proposed approach uses auto-regressive moving average modeling to forecast the future temperature. A proactive dynamic thermal management is proposed in [128] based on predict-and-act philosophy. In the proposed framework, the operating system scheduler predicts the temperature of the individual cores; if the temperature of

Table 2.4: Related works on run-time reliability optimization techniques.

Related Works	**DRM Approach**	**Workload Variation**	**Thermal Cycling**	**Temperature / Reliability Measurements**	**Validation Platform**
Karl et. al. [122], Hanumaiah et al. [123], Faruque et al. [124]	heuristic	intra variation	×	thermal model	multicore
Hartman et al. [125]	heuristic	intra variation	×	wear-out sensors	simulation
Chantem et al. [126]	heuristic	intra variation	√	thermal model	simulation
Coskun et al. [127], Ayoub et al. [128], Cochran et al. [129]	predict-and-act	intra variation	×	thermal sensors	multicore
Sironi et al. [130]	observed-decide-act	intra variation	×	thermal sensors	multicore
Bolchini et al. [131]	observe-decide-act	intra variation	×	*HotSpot*	simulation
Mercati et al. [132]	observe-decide-act	intra variation	√	wear-out sensors	multicore
Lee et al. [133]	machine learning	intra variation	×	sensors	multicore
Jayaseelan et al. [134]	machine learning	intra variation	×	*HotSpot*	simulation
Ge et al. [135]	machine learning	intra variation	×	thermal models & sensors	multicore
Ebi et al. [136]	machine learning	intra variation	×	thermal gun	*FPGA*
Objectives of this thesis [137]	machine learning	inter & intra variation	√	thermal sensors	multicore

a core crosses a pre-defined value, the operating system decides to migrate one or more threads of a given workload to the coldest core in the system. A thermal management technique is proposed in [129], which predicts the workload phase change and selects the appropriate voltage and frequency of the cores to minimize the peak temperature.

A discrete-time thermal model is proposed in [130] for dynamic thermal management. The proposed technique monitors the temperature sensors and decides on the length of the idle time needed to reduce thermal emergencies. An adaptive approach is proposed in [131] to minimize the electromigration-related wear-out by monitoring the aging at runtime and controlling it through task mapping. A control theoretic approach is proposed in [132] to maximize the lifetime of homogeneous multicore systems. The proposed approach is based on long term controller, which samples data from aging sensors to compute the wear-out degradation. Based on this, the short-term controller adjusts the voltage and frequency of the tasks to minimize temperature while satisfying the performance requirement and the user experience.

A machine learning technique is proposed in [133] to dynamically manage the peak temperature for MPEG2 video decoding. The technique is very application specific and can be applied only to video decoding applications such as MPEG4 or H.264. A neural network-based adaptive thermal management policy is proposed in [134]. The technique relies on temperature prediction using the *HotSpot* tool. A reinforcement learning algorithm is proposed in [135] to manage performance-thermal trade-offs by sampling temper-

ature data from the on-board thermal sensors. A distributed learning agent is proposed in [136] to optimize peak temperature within a given power budget. The technique is implemented on FPGA with temperature measurement using an external thermal gun. Table 2.4 summarizes these related works and highlights the objectives of this thesis.

2.5 Remarks

This chapter provided an overview of the operational semantics of the SDFG that is used in the subsequent chapters as the application model. The chapter also provided the background for reliability modeling of multiprocessor systems considering different device-level wear-out mechanisms and their dependency on temperature. Finally, a detailed background study is provided on the existing system-level design techniques for lifetime optimization, highlighting the objectives achieved in this thesis.

CHAPTER 3

Reliability and Energy-Aware Platform-Based Design Methodology

3.1 Introduction

As discussed in Chapter 2, a significant research attention is directed recently to investigate platform-based design approaches in order to mitigate wear-out and minimize energy consumption. These studies, however, suffer from the following two limitations – accuracy and scope.

Accuracy: Most existing studies on thermal and reliability management ignore either the transient phase of the temperature or the spatial dependency. Ignoring the transient temperature simplifies the thermal analysis, but is accurate if the execution times of the tasks are comparable to the thermal time constant of the package, which is typically of the order of few hundreds of seconds. Finally, ignoring the spatial dependency leads to simplification of the RC equivalent model, but results in underestimation of the temperature and a corresponding overestimation of the mean time to failure. Additionally, some existing studies on lifetime reliability approximate MTTF as the time to the first fault. This is true for systems that are not provisioned to tolerate faults. In this work, multiprocessor systems are considered with support for task migration. Such a system continues

to operate in the presence of failures, albeit an acceptable performance degradation. For such systems, estimating the MTTF as the time to first failure leaves a significant scope of improvement, both in terms of lifetime and energy consumption.

Scope: The existing lifetime optimization techniques are all based on sequential execution of applications represented as directed acyclic graphs (DAGs). Synchronous data flow graphs (SDFGs) allow more suitable modeling for streaming multimedia and other data flow applications that require support for multi-input tasks, multi-rate tasks, pipelined execution and a natural way for dealing with latency and buffer requirements.. The existing techniques for DAGs cannot be applied directly on SDFGs due to the cyclic actor dependencies and the overlapping of multiple iterations (pipelined) in the schedule. Furthermore, all the existing techniques determine lifetime optimum mapping for a single application. However, multiprocessor systems are often designed for multiple applications, many of them enabled concurrently (use-case). As shown in this work, lifetime-aware distribution of the cores among the concurrent applications leads to a significant improvement in MTTF.

To address the limitations of the existing approaches, a temperature model is first proposed that is based on off-line thermal characterization of a multiprocessor system using the *HotSpot* tool. The model incorporates the following:

1. *temporal dependency* i.e., the relationship between the temperature of a core as a function of time and its dependency on the operating voltage and frequency; and

2. *spatial dependency* i.e., the influence of the neighboring core's temperature on the temperature of a core.

A gradient-based fast heuristic is then proposed incorporating the temperature model, to jointly optimize energy and lifetime reliability of a multiprocessor system with applications represented as SDFGs. The approach leverage on the native SDF^3 tool [138], and can be easily ported to applications represented as DAGs, making the approach generic for both streaming multimedia and non-multimedia applications. Following are the key contributions of this work:

- a simplified temperature-model considering temporal and spatial dependencies;

- computing the MTTF considering task remapping;

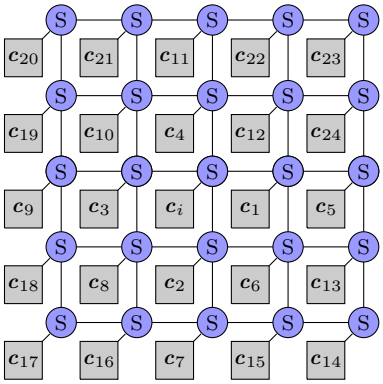


Figure 3.1: Multiprocessor floorplan.

- a gradient-based fast heuristic to jointly optimize lifetime reliability and energy;
- reliability optimization considering synchronous data flow graphs; and
- MTTF maximization considering single and multi-application use-cases.

The remainder of this chapter is organized as follows. The problem formulation is discussed in Section 3.2 followed by the proposed temperature model in Section 3.3. The temperature computation from a given SDFG schedule is demonstrated in Section 3.4. The design methodology is discussed next in Sections 3.5. Experimental results are presented in Section 3.6 and Section 3.7 presents the conclusions.

3.2 Problem Formulation

3.2.1 Application Model

An application SDFG is mathematically represented as $\mathcal{G}_{app} = (\mathbb{A}, \mathcal{C})$ consisting of a finite set $\mathbb{A}$ of actors and a finite set $\mathcal{C}$ of channels. Every actor $\boldsymbol{a}_i \in \mathbb{A}$ s a tuple (t_i, μ_i), where t_i is the execution time of $\boldsymbol{a}_i$ and μ_i is its state space (program and data memory). The number of actors in an SDFG is denoted by N_a where $N_a = |\mathbb{A}|$. The performance of an SDFG is specified in terms of throughput constraint $\mathbb{T}_c$.

3.2.2 Architecture Model

The multiprocessor architecture for this work is shown in Figure 3.1 with cores (indicated as labeled boxes) interconnected through switches in a mesh-based topology (the labels on the cores are explained in Section 3.3). This model is a representative of a single chip

multi-/many-core system such as Tilera [18].[1]

For problem formulation, the adopted architecture is represented as a graph $\mathcal{G}_{arc} = (\mathbb{C}, \mathbb{E})$, where $\mathbb{C}$ is the set of nodes representing cores of the architecture and $\mathbb{E}$ is the set of edges representing communication channels among the cores. The number of cores in the architecture is denoted by N_c i.e., $N_c = |\mathbb{C}|$. Each core $\boldsymbol{c}_j \in \mathbb{C}$ supports N_f voltage-frequency pairs denoted by $\{(V_k, \omega_k) \ \ \forall k \in [0, N_f - 1]\}$.

3.2.3 Mapping Representation

The objective of the optimization problem is to maximize the lifetime reliability (measured as MTTF) and minimize the energy consumption by solving the following:

- *actor distribution:* determine the assignment of the actors of the SDFG on the cores of the multiprocessor system;
- *operating point:* determine the voltage and frequency of the cores for executing the actors of the SDFG.

For the ease of problem representation, two variables $x_{i,j}$ (representing the actor distribution) and $y_{i,k}$ (representing the operating point) are defined as follows.

$$x_{i,j} = \begin{cases} 1 & \text{if actor } \boldsymbol{a}_i \text{ is executed on core } \boldsymbol{c}_j \\ 0 & \text{otherwise} \end{cases}$$

$$y_{i,k} = \begin{cases} 1 & \text{if actor } \boldsymbol{a}_i \text{ is executed at operating point } (V_k, \omega_k) \\ 0 & \text{otherwise} \end{cases}$$

Constraints on these variables are set such that an actor is mapped to only one core at a single operating point. Thus,

$$\sum_{j=0}^{N_c-1} x_{i,j} = 1 \text{ and } \sum_{k=0}^{N_f-1} y_{i,k} = 1 \quad \forall \boldsymbol{a}_i \in \mathbb{A} \tag{3.1}$$

The actor distribution and operating point of SDFG are represented as two matrices:

$$\mathcal{M}_d = \begin{pmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,N_c-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,N_c-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N_a-1,0} & x_{N_a-1,1} & \cdots & x_{N_a-1,N_c-1} \end{pmatrix} \tag{3.2}$$

[1]The technique proposed in this thesis can also be adopted for other widely-accepted architectures such as those involving multi-chip systems by interfacing with the operating system's thread scheduling and affinity mapping functions.

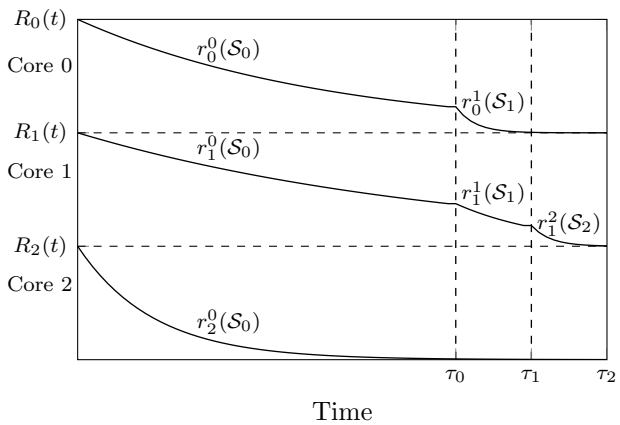


Figure 3.2: MTTF computation with different temperature profile.

$$\mathcal{M}_o = \begin{pmatrix} y_{0,0} & y_{0,1} & \cdots & y_{0,N_f-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,N_f-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N_a-1,0} & y_{N_a-1,1} & \cdots & y_{N_a-1,N_f-1} \end{pmatrix} \tag{3.3}$$

The core to which actor $\boldsymbol{a}_i$ is assigned is denoted by ϕ_i and is given by $\phi_i = \mathbf{X}_i \times \mathbb{N}_{N_c}$ where $\mathbf{X}_i = \begin{pmatrix} x_{i,0} & x_{i,1} & \cdots & x_{i,N_c-1} \end{pmatrix}$ and $\mathbb{N}_{N_c}$ is the matrix of integers from 0 to N_c i.e., $\mathbb{N}_{N_c} = \begin{pmatrix} 0 & 1 & \cdots & N_c - 1 \end{pmatrix}^T$. The operating point of actor $\boldsymbol{a}_i$ is denoted by θ_i and is given by $\theta_i = \mathbf{Y}_i \times \mathbb{N}_{N_f}$ where $\mathbf{Y}_i = \begin{pmatrix} y_{i,0} & y_{i,1} & \cdots & y_{i,N_f-1} \end{pmatrix}$.

3.2.4 MTTF Computation

To demonstrate the MTTF computation using the iterative approach (Chapter 2), an example is provided with three cores. The initial schedule $\mathcal{S}_0$ uses all the three cores and stresses core 2 more than the other two cores. The reliability curves for the three cores are shown in Figure 3.2. Core 2 has the least lifetime and it breaks at time τ_0. As indicated in Chapter 2, most existing works on lifetime reliability defines MTTF to be the time to the first failure, hence the MTTF for these works is τ_0. At time $t = \tau_0$, a second schedule $\mathcal{S}_1$ (using core 0 and 1) is applied. The change in the reliability profile of core 0 and core 1 are due to different wear-out, which can be attributed to the difference in temperature from this new schedule. The new schedule stresses core 0 more than core 1 and therefore core 0 breaks at time $t = \tau_1$. At this time, all the actors are remapped to core 1 and are ordered honoring the actor dependency. This schedule is identified in the figure as $\mathcal{S}_2$ and results in reliability profile of r_1^2. With this new reliability profile, core 1 breaks at time $t = \tau_2$. The lifetime (MTTF) of the system is therefore τ_2.

Besides MTTF, another interesting metric for multiprocessor systems supporting task remapping is the *processor years*, defined as the aggregate utilization of the different cores of the system over the entire lifetime. For the above example, this is calculated as follows. For the interval 0 to τ_0, all three cores are active; for the interval τ_0 to τ_1 two core are active; and for the interval τ_1 to τ_2 only one core is active. Assuming the time in this figure are all in years, the *processor years* of the above system is $3 \cdot \tau_0 + 2 \cdot (\tau_1 - \tau_0) + 1 \cdot (\tau_2 - \tau_1)$.

3.2.5 Energy Modeling

Actor-level voltage and frequency scaling is assumed for this work i.e, every actor of an SDFG is associated with a voltage-frequency value that is set on the core executing the actor. The energy consumed on a multiprocessor system while executing the SDFG consists of the following components:

- *computation energy:* dynamic and leakage energy consumed on the cores due to the actors execution; and
- *communication energy:* dynamic and leakage energy consumed on the network-on-chip (NoC) due to the data communication among the connected actors.

A point to note here is that the leakage energy consumed on the NoC is dependent on the NoC type and the topology. For this work, a spatial division multiplexing-based NoC is assumed and therefore, the leakage power consumed on the NoC is negligible [139].

Dynamic Energy of SDFG: The dynamic energy of an SDFG is given by $E^{dyn} = E^{dyn}_{tr} + N_{iter} \cdot E^{dyn}_{ss}$, where E^{dyn}_{tr} is the actor dynamic energy in the transient phase of the schedule, E^{dyn}_{ss} is the actor dynamic energy per iteration of the steady state phase and N_{iter} is the number of iterations of the steady state phase. Usually, the number of steady state iterations (i.e.,N_{iter}) is a large number (e.g., periodic decoding of video frames) and hence for all practical purposes, the dynamic energy of the steady state phase dominates over that in the transient phase. Throughout the rest of this chapter, computation (or communication) energy implies computation (or communication) energy of the steady state phase per iteration.

The dynamic energy consumed by an actor $\boldsymbol{a}_i$ executed on core $\boldsymbol{c}_j$ at the operating point k is given by

$$e^{dyn}(i,j,k) = C_{eff} \cdot \beta \cdot V_k^2 \cdot \omega_k \cdot t_{ijk} \cdot RV[\boldsymbol{a}_i] \tag{3.4}$$

where β is the activity factor, C_{eff} is the effective load capacitance, t_{ijk} is the execution time of actor $\boldsymbol{a}_i$ on core $\boldsymbol{c}_j$ at operating point k (i.e.,operating voltage V_k and operating frequency ω_k) and $RV[\boldsymbol{a}_i]$ is the number of firings of actor $\boldsymbol{a}_i$ per steady state iteration of the SDFG. The total dynamic energy per steady-state iteration of the SDFG is

$$E_{core}^{dyn} = \sum_{\forall \boldsymbol{a}_i \in \mathbb{A}} e^{dyn}(i, \phi_i, \theta_i) \tag{3.5}$$

Leakage Energy of SDFG: The leakage energy of core $\boldsymbol{c}_j$, consumed during the execution of actor $\boldsymbol{a}_i$ at operating point k, is given by the following formula [140].

$$e^{leak}(i, j, k) = N_{gates} V_k I_0 \left[A T^2 e^{\frac{\alpha V_k + \beta}{T}} + B e^{\gamma V_k + \delta} \right] \cdot t_{ijk} \cdot RV[\boldsymbol{a}_i] \tag{3.6}$$

where N_{gates} is the number of gates of the core, I_0 is the average leakage current and $A, B, \alpha, \beta, \gamma, \delta$ are technology dependent constants (refer to [140]) and T is the average temperature of the actor during the steady-state iteration. The total leakage energy is

$$E_{core}^{leak} = \sum_{\forall \boldsymbol{a}_i \in \mathbb{A}} e^{leak}(i, \phi_i, \theta_i) \tag{3.7}$$

Dynamic Energy on the NoC: In [41], bit energy (E_{bit}) is defined as the energy consumed in transmitting one bit of data through the routers and links of a NoC.

$$E_{bit} = E_{S_{bit}} + E_{L_{bit}} \tag{3.8}$$

where $E_{S_{bit}}$ and $E_{L_{bit}}$ are the energy consumed in the switch and the link, respectively. The energy per bit consumed in transferring data between cores $\boldsymbol{c}_p$ and $\boldsymbol{c}_q$, situated $n_{hops}(p, q)$ away is given by Equation 3.9 according to [41].

$$E_{bit}(p, q) = \begin{cases} (n_{hops}(p, q) + 1) \cdot E_{S_{bit}} + n_{hops}(p, q) \cdot E_{L_{bit}} & \text{if } p \neq q \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

The dynamic energy consumed on the NoC is therefore given by Equation 3.10 where ϕ_i and $\phi_{i'}$ are the cores where actors $\boldsymbol{a}_i$ and $\boldsymbol{a}_{i'}$ are mapped, respectively.

$$E_{noc}^{dyn} = \sum_{\forall \boldsymbol{a}_i, \boldsymbol{a}_{i'} \in \mathbb{A}} d_{ij} \cdot E_{bit}(\phi_i, \phi_{i'}) \tag{3.10}$$

The total energy is

$$E^{tot} = E_{core}^{dyn} + E_{core}^{leak} + E_{noc}^{dyn} \tag{3.11}$$

3.2.6 Reliability-Energy Joint Metric

To jointly optimize lifetime reliability and energy, a single metric **lifetime quotient** (lq) is introduced, which as defined as the ratio of the MTTF to the total energy i.e.,

$$lq = \frac{MTTF}{E^{tot}} \tag{3.12}$$

The optimization objective is to maximize lq.

3.3 Proposed Temperature Model

Temperature of a multiprocessor system is usually determined using a RC equivalent thermal model. The temperature of a single core is related to its power dissipation according to the following equation [68].

$$C\frac{dT(t)}{dt} + G\left(T(t) - T_{amb}\right) = P(t) \tag{3.13}$$

where C is the thermal capacitance, G is the thermal conductance, t is the time, T_{amb} is the ambient temperature, $T(t)$ is the instantaneous temperature and $P(t)$ is the instantaneous power that is composed of the dynamic and the static components. The dynamic power is dependent on the voltage and frequency of operation and the static power is dependent on the temperature (refer to Section 3.2.5). The solution to the above equation consists of transient and steady-state phases. In the transient phase, the temperature increases with time up to a point beyond which, the steady-state phase settles in and the temperature saturates to its steady-state value. The core's wear-out is dependent on its operating temperature, which needs to incorporate both the transient and the steady-state phases. For a multiprocessor system with interconnected cores (refer to Figure 3.1), the temperature of any core, say core $\boldsymbol{c}_i$ depends on

A.1 The time of execution of an actor on $\boldsymbol{c}_i$.

A.2 The voltage and frequency of $\boldsymbol{c}_i$.

A.3 The temperature of the cores surrounding $\boldsymbol{c}_i$.

Thus, A.1 and A.2 represents the temporal dependency and A.3 represents the spatial dependency. For such a system, the temperature, power, thermal capacitance and thermal conductance in Equation 3.13 are all vectors. The transient and steady-state values can be obtained by solving the above equation analytically. The solution to the differential equation is

$$\mathbf{T}(t) = e^{\kappa t}\mathbf{T}(0) + \kappa^{-1}\left(e^{\kappa t} - \mathbf{I}\right)\mathbf{C}^{-1}\mathbf{P}\left(t\right) \tag{3.14}$$

where $\kappa = -\mathbf{C}^{-1}\mathbf{G}$, $\mathbf{T}(0)$ is the initial temperature and $\mathbf{I}$ is the identity matrix. The direct solution techniques, such as LU decomposition and sparse solver, are usually slow and results in an exponential design space exploration time. Although the iterative

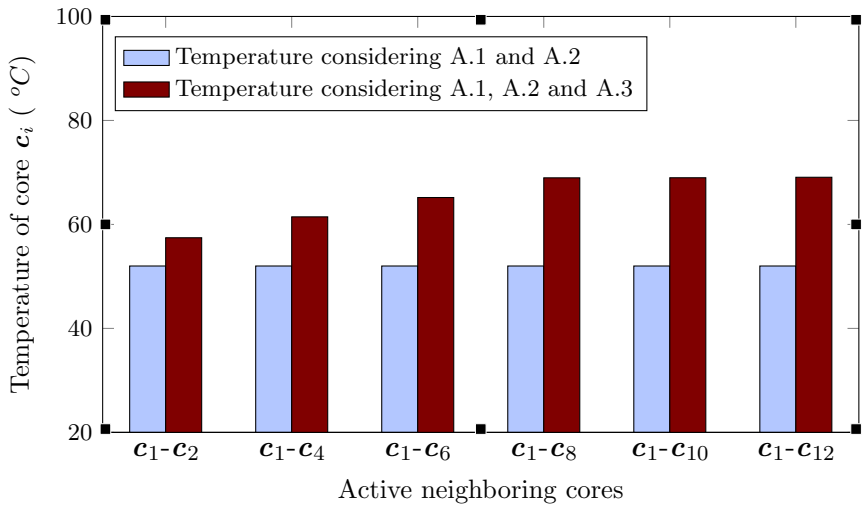


Figure 3.3: Temperature underestimation ignoring the spatial dependency.

technique of [97] simplifies the solution, the execution time is still exponential when applied to multi-application use-cases.

A simplification to the analytical approach is to ignore the spatial dependency by considering the temperature for cores individually, such as the one proposed in [141]. To signify the importance of temperature underestimation by ignoring the spatial dependency (component A.3), an experiment is conducted using the *HotSpot* tool to measure the steady-state temperature. The multiprocessor architecture used for the *HotSpot* tool is shown in Figure 3.1 with the specifications of the cores reported in Table 3.1. The temperature is determined by setting the power dissipation of core $\boldsymbol{c}_i$ as 0, with a constant power corresponding to the operating point OPP1G (i.e.,1.35V, 1GHz) set for the one-hop and the two-hop neighboring cores (cores $\boldsymbol{c}_1$ - $\boldsymbol{c}_4$ are the one-hop neighbors and cores $\boldsymbol{c}_5$ - $\boldsymbol{c}_{12}$ are the two-hop neighbors of core $\boldsymbol{c}_i$ in Figure 3.1). This simulates the scenario of core $\boldsymbol{c}_i$ as idle with the neighboring cores active at the highest voltage and frequency. The temperature results are reported in Figure 3.3 for some combinations of the neighboring core's activity. The label $\boldsymbol{c}_1 - \boldsymbol{c}_n$ in the figure indicates cores $\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_n$ are active simultaneously. There are two bars shown on the plot. The left bar for each label corresponds to the temperature of core $\boldsymbol{c}_i$ obtained with all the core as idle. The right bar corresponds to the temperature of $\boldsymbol{c}_i$ with cores $\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_n$ operating at the highest operating point and core $\boldsymbol{c}_i$ as idle. As seen from the figure, with only the east and the south neighbors active (i.e.,label $\boldsymbol{c}_1 - \boldsymbol{c}_2$), the temperature considering spatial dependency is 5°C higher than the temperature obtained by ignoring the spatial dependency (i.e., A.3). This difference increases as more number of neighbors become active.

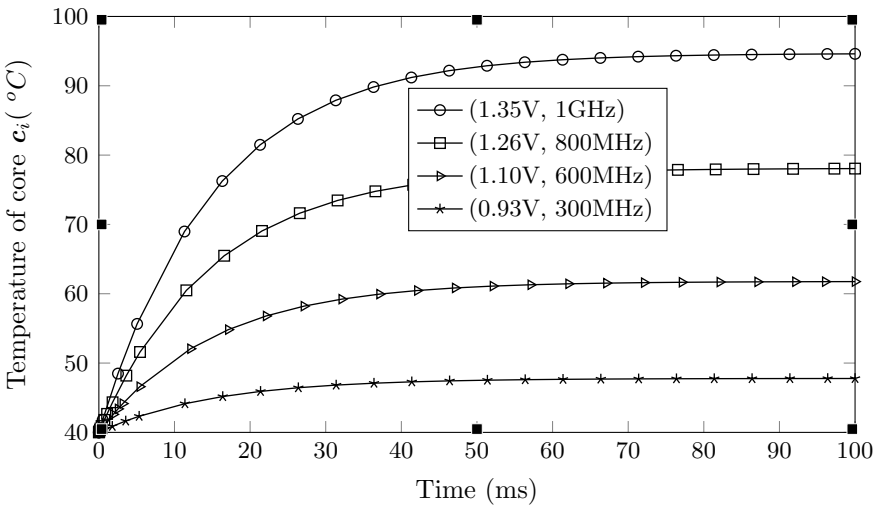


Figure 3.4: Characterization for temporal dependency.

Finally, with all the one-hop and two-hop neighbors active, the temperature difference is as high as 18°C. This temperature underestimation leads to MTTF misprediction.

To provide a simplified solution of Equation 3.14, a regression analysis technique is proposed in this work. The proposed temperature model is based on:

- temperature characterization to incorporate the temporal dependency (capturing both the transient and steady-state behaviors); and
- temperature characterization to incorporate the spatial temperature dependency.

The solution to the differential equation is represented as

$$T_i(t) = f(V_i, \omega_i, t) + g(\{V_j, \omega_j \mid \forall \boldsymbol{c}_j \in \aleph(\boldsymbol{c}_i)\}) \tag{3.15}$$

where (V_i, ω_i) are the voltage and frequency of core $\boldsymbol{c}_i$, t is the time and $\aleph(\boldsymbol{c}_i)$ are the cores in the neighborhood of $\boldsymbol{c}_i$. The function f and g represent the temporal and the spatial dependency, respectively and are derived in two steps.

- **Determine f:** The function f can be determined using two alternative approaches – by solving Equation 3.13 directly for a processing core; or by simulation using the *HotSpot* tool for the power consumption corresponding to different operating points of the processing core to capture the transient and the steady-state behaviors, as shown in Figure 3.4.
- **Characterize g:** The temperature data for characterizing the function g are obtained as follows. The core $\boldsymbol{c}_i$ is set to idle mode and the operating points for the

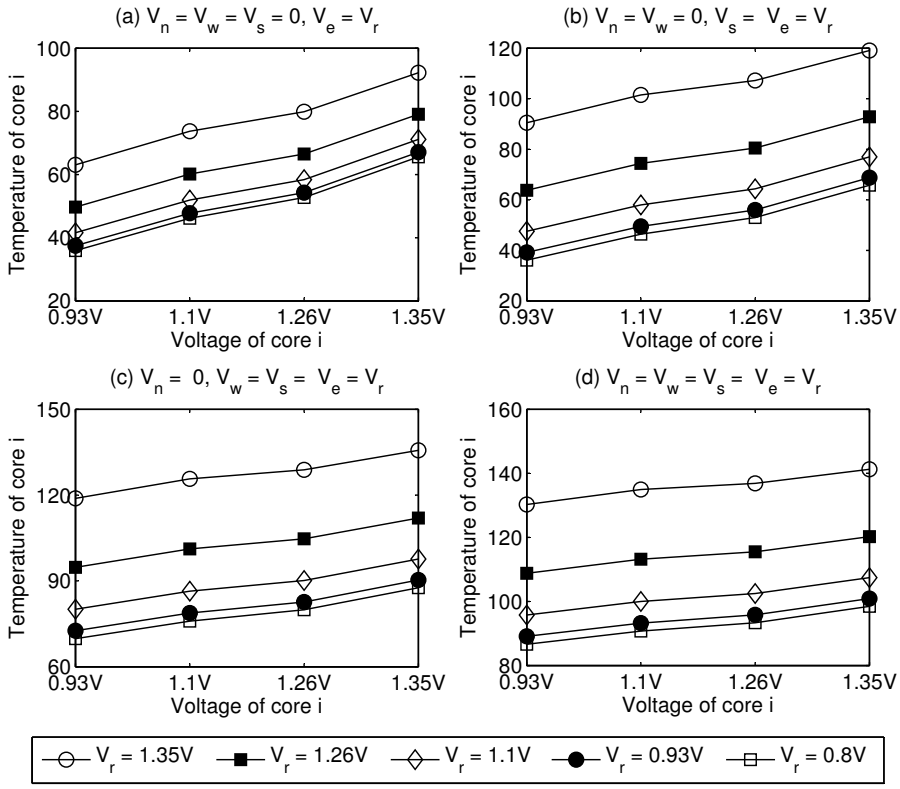


Figure 3.5: Characterization for spatial dependency.

neighboring cores are varied. Performing exhaustive temperature simulations for different voltage-frequency combinations of all neighbors (one-hop neighbors, two-hop neighbors, etc.) is time consuming, but only required once during the characterization step. A first order of approximation involves considering the voltage-frequency of only the immediate neighbors i.e., the east, west, north and south neighbors of a core referred to as (V_e, ω_e), (V_w, ω_w), (V_n, ω_n) and (V_s, ω_s), respectively with all other neighbors set to operate at the highest operating point. This is shown in Figure 3.5, where the voltage point is only shown for clarity of representation. The figure plots the temperature of core $\boldsymbol{c}_i$ as its voltage V_i is increased from 0.93V to 1.35V for few of these neighboring voltage combinations.

The temperature data are fed to the Matlab regression toolbox to derive the temperature model. The proposed temperature model is determined once during the characterization process. The final expression for temperature (Equation 3.15) can be easily integrated in the design space exploration framework. However, the proposed model incorporates pessimism in three forms – separating the temporal and spatial dependency; characterizing the spatial dependency with the steady-state temperature of the nearest neighbors;

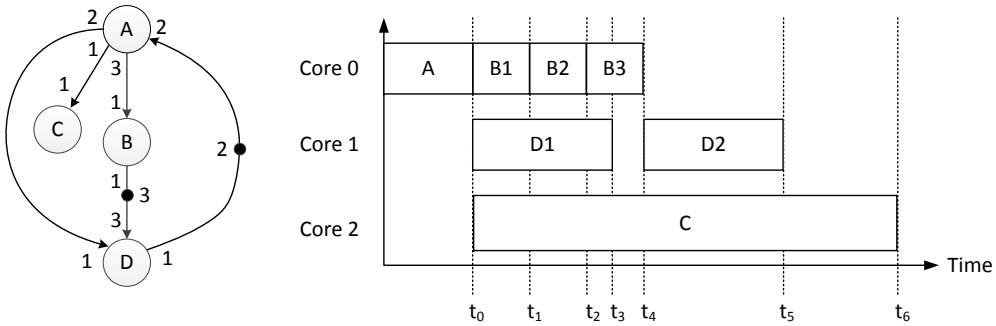


Figure 3.6: Temperature computation from an SDFG schedule.

and characterizing the spatial dependency with the non-nearest neighbors set to operate at the highest voltage and frequency. These pessimism lead to a temperature overestimation by as much as up to 6°C for individual applications. However, as discussed in Section 3.5.2, this temperature overestimation simplifies the reliability optimization for multi-application use-cases, which is a common requirement for multiprocessor systems.

3.4 Temperature Computation from a Schedule

Figure 3.6 shows an example of an SDFG with four actors allocated on a platform with three cores. The schedule corresponding to a particular allocation is also shown in the same figure. The temperature computation is demonstrated for core 0 using this schedule. The temperature for other cores can be determined in a similar fashion. The time duration $0 - t_6$ is divided into seven intervals by putting a time stamp at the instances where the actors start or end *firing*.

Core 0: Interval $(0 \rightarrow t_0)$

In this interval, core 0 executes actor A at operating point (V_A, ω_A). The temperature at time t considering the temporal effect is $f(V_A, \omega_A, t)$. The temperature considering the spatial effect is due to the idle voltages of core 1 and 2 and is given by $g(V_{idle}, V_{idle})$. The average temperature in this interval is

$$T_0(0, t_0) = \frac{1}{t_0} \int_0^{t_0} f(V_A, \omega_A, t)dt + g(V_{idle}, V_{idle}) \tag{3.16}$$

Core 0: Interval $(t_0 \rightarrow t_1)$

In this interval, core 0 executes the first instance of actor B. Note in the SDFG, when actor A fires, it produces 3 tokens on the channel from actor A to actor B and one of these tokens is consumed for each firing of actor B. Therefore, there are three firing of actor B (indicated in the figure by B1, B2 and B3). The temperature at time t due to

the temporal effect of actor B is $f(V_B, \omega_B, t)$ and the temperature due to spatial effect is $g(V_D, V_C)$. The average temperature is

$$T_0(t_0, t_1) = \frac{1}{t_1 - t_0} \int_0^{t_1 - t_0} f(V_B, \omega_B, t)dt + g(V_D, V_C) \tag{3.17}$$

Core 0: Interval $(t_1 \rightarrow t_2)$

The temperature computation in this interval is similar to that in the interval $(t_0 \rightarrow t_1)$ and is given by

$$T_0(t_1, t_2) = \frac{1}{t_2 - t_1} \int_0^{t_2 - t_1} f(V_B, \omega_B, t)dt + g(V_D, V_C) \tag{3.18}$$

Core 0: Interval $(t_2 \rightarrow t_3)$

During the execution of actor B3, there is a change in temperature profile due to the completion of actor D1 and the interval before actor D2 is executed. Hence, the execution time of actor B3 is split into two intervals $(t_2 \rightarrow t_3)$ and $(t_3 \rightarrow t_4)$. The temperature computation in the interval $(t_2 \rightarrow t_3)$ is similar to that in the interval $(t_0 \rightarrow t_1)$

$$T_0(t_2, t_3) = \frac{1}{t_3 - t_2} \int_0^{t_3 - t_2} f(V_B, \omega_B, t)dt + g(V_D, V_C) \tag{3.19}$$

Core 0: Interval $(t_3 \rightarrow t_4)$

The average temperature in this interval is given by

$$T_0(t_3, t_4) = \frac{1}{t_4 - t_3} \int_0^{t_4 - t_3} f(V_B, \omega_B, t)dt + g(V_{idle}, V_C) \tag{3.20}$$

Core 0: Interval $(t_4 \rightarrow t_5)$

In this interval, the temporal effect is due to the idle temperature of the core and is denoted by T_0^{idle}. The average temperature is given by

$$T_0(t_4, t_5) = T_0^{idle} + g(V_D, V_C) \tag{3.21}$$

Core 0: Interval $(t_5 \rightarrow t_6)$

The temperature in this interval is given by

$$T_0(t_5, t_6) = T_0^{idle} + g(V_{idle}, V_C) \tag{3.22}$$

Reliability of Core 0

Combining these equations, the aging of core 0 is

$$r_0 = \frac{1}{t_6} \sum_{i=0}^{6} \frac{t_i - t_{i-1}}{\alpha\left(T_0(t_i, t_{i-1})\right)} \tag{3.23}$$

where $t_{-1} = 0$ and α is the fault density.

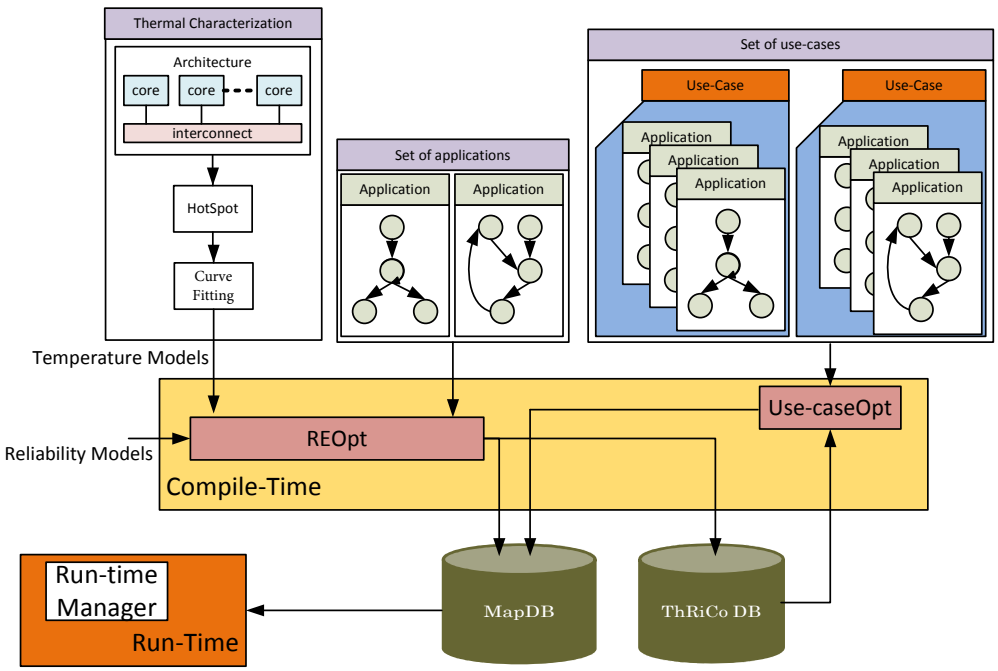


Figure 3.7: Proposed design methodology.

3.5 Design Methodology

The design methodology consists of two phases – analysis at design-time (consisting of the application and use-case optimizations) and execution at run-time. The design-time methodology is highlighted in Figure 3.7. The run-time manager is not part of the contribution, but is shown here for completeness. There are two databases for the multiprocessor system – the set of applications ($\mathbf{S}_{app}$) and the set of use-cases ($\mathbf{S}_{use}$). The proposed approach is to determine the actor distribution and the operating point (refer to Section 3.2) for every application using $n = N_c^{min}$ to N_c cores of the system. Thus, $|\mathbf{S}_{app}| \cdot (N_c - N_c^{min} + 1)$ optimization problems are solved at design-time. This is performed in the *REOpt* block. The solution consists of the actor distribution and operating point matrices stored in the *MapDB* database and the three-dimensional (3D) vector – throughput, reliability (MTTF) and core count stored in the *ThRiCoDB* database.

Algorithm 2 provides the pseudo-code of the design flow. For every application $\mathbf{A}_i$ of the set $\mathbf{S}_{app}$, the corresponding SDFG representation and the throughput constraint are fetched from the database. This application is executed on the multiprocessor system with n cores identified as $\mathcal{G}_{arc}^n$, where n is varied from N_c^{min} to N_c. The reliability-energy joint optimization is first performed on the application (line 5) to obtain the actor distribution matrix $\mathcal{M}_d$ and the operating point matrix $\mathcal{M}_o$. These are stored in

Algorithm 2 Generate reliability and energy aware mappings.

Input: Application set $\mathbf{S}_{app}$ and multiprocessor system $\mathcal{G}_{arc}$
Output: $MapDB$ and $ThRiCoDB$

1: **for all** Application $\mathbf{A}_i \in \mathbf{S}_{app}$ **do**
2: $[\mathcal{G}_{app} \;\; \mathbb{T}_c] = GetSDFG(\mathbf{A}_i)$ *//Get the corresponding SDFG and the throughput constraint.*
3: Determine N_c^{min}, the minimum number of cores required for satisfying the throughput requirement.
4: **for** $n = N_c^{min}$ to N_c **do**
5: $\begin{pmatrix}\mathcal{M}_d & \mathcal{M}_o\end{pmatrix} = REOpt(\mathcal{G}_{app}, \mathcal{G}_{arc}^n, \mathbb{T}_c)$ *//Perform individual application optimization*
6: $[\mathcal{S} \;\; \mathbb{T}] = MSDF^3(\mathcal{M}_d, \mathcal{M}_o, \mathcal{G}_{app}, \mathcal{G}_{arc}^n)$ *//Calculate the schedule and the throughput of the SDFG.*
7: $MapDB(i,n) = \begin{pmatrix}\mathcal{M}_d & \mathcal{M}_o & \mathcal{S}\end{pmatrix}$ *//Store the mapping in the mapping database.*
8: $M = CalculateMTTF(i, n, N_c^{min}, MapDB)$
9: $ThRiCoDB(i,n) = \begin{pmatrix}\mathbb{T} & M\end{pmatrix}$ *//Store the throughput and MTTF values for the use-case optimization step.*
10: **end for**
11: **end for**

Algorithm 3 $CalculateMTTF()$: Calculate the mean time to failure.

Input: Application id i, the core index n, the minimum number of cores for throughput satisfaction and mapping database $MapDB$
Output: MTTF M

1: Initialize $ttf = 0$ and $ri = n$
2: **while** $ri \geq N_c^{min}$ **do**
3: $[\mathcal{M}_d \;\; \mathcal{M}_o \;\; \mathcal{S}] = MapDB(i, ri)$ *//Fetch the values.*
4: Determine reliability profiles from $\mathcal{S}$ as demonstrated in Section 3.4
5: Shift the reliability profiles by ttf
6: Determine t, the time to failure of the most stressed core
7: $ttf = ttf + t$ and $ri = ri - 1$
8: **end while**

the $MapDB$ (line 7). The actor distribution is used in the $MSDF^3$ tool that leverage on the native SDF^3 tool [138][2] to generate the throughput and schedule. The schedule thus obtained is used in the $CalculateMTTF()$ routine (line 8) to compute the MTTF. The throughput and the MTTF values corresponding to the number of cores are stored in the $ThRiCoDB$ for the use-case optimization step that addresses core distribution among concurrent applications.

The $CalculateMTTF$ routine determines the MTTF in an iterative manner as shown as pseudo-code in Algorithm 3. A running index ri is maintained to index to the schedule with one less core. At the start of the iteration, the mapping and the scheduling are fetched from the $MapDB$. The schedule is used to compute the reliability profile of every core of the system. The reliability profiles are shifted to account for the aging already encountered in the cores. The time-to-failure for all the cores are determined using Equation 2.21. The minimum time corresponds to the failure of the most stressed core. This is added to the ttf and the running index is decremented.

[2]The native SDF^3 tool generates one feasible actor distribution and the corresponding throughput. The $MSDF^3$ tool is modified form of SDF^3 that generates the schedule and throughput from a given actor distribution matrix.

Algorithm 4 *REOpt*(): Reliability and energy optimization for an application.

Input: $\mathcal{G}_{app}$, $\mathcal{G}_{arc}$ and throughput constraint $\mathbb{T}_c$
Output: actor distribution and operating point matrices $\begin{pmatrix}\mathcal{M}_d & \mathcal{M}_o\end{pmatrix}$, which maximize lq

1: Initialize $\mathcal{M}_o = \begin{pmatrix}\mathbf{0} & \mathbf{0} & \cdots & \mathbf{1}\end{pmatrix}$ *//Initialize the actors to the highest operating point.*
2: $[\mathcal{M}_d \;\; \mathcal{S} \;\; \mathbb{T}] = SDF^3(\mathcal{G}_{app}, \mathcal{G}_{arc})$ *//Mapping, schedule and throughput using the native SDF^3 tool.*
3: **while** true **do**
4: $\quad \mathcal{P}^{best} = 0$, $\mathcal{M}_d^{best} = \mathcal{M}_d$, $best\_found = false$ *//Initialize the best values.*
5: $\quad lq = CalculateLQ(\mathcal{M}_d, \mathcal{M}_o, \mathcal{S}, \mathbb{T})$ *//Calculate the initial lifetime quotient.*
6: $\quad$ **for all** $\boldsymbol{a}_i \in \mathbb{A}$ **do**
7: $\quad\quad$ **for all** $\boldsymbol{c}_j \in \mathbb{C}$ **do**
8: $\quad\quad\quad$ **for all** $k \in [0, N_f - 1)$ **do**
9: $\quad\quad\quad\quad \mathcal{M}_d^{temp} = \mathcal{M}_d$ and $\mathcal{M}_o^{temp} = \mathcal{M}_o$ *//A temporary allocation matrix is used.*
10: $\quad\quad\quad\quad$ Update $\mathcal{M}_d^{temp}, \mathcal{M}_o^{temp}$ using $x_{i,j} = y_{i,k} = 1$ and $x_{i,l} = y_{i,m} = 0$, $\forall l \neq j$ and $\forall m \neq k$
11: $\quad\quad\quad\quad [\mathcal{S}_n \;\; \mathbb{T}_n] = MSDF^3(\mathcal{M}_d^{temp}, \mathcal{M}_o^{temp}, \mathcal{G}_{app}, \mathcal{G}_{arc})$ *//New schedule is computed.*
12: $\quad\quad\quad\quad lq_n = CalculateLQ(\mathcal{M}_d^{temp}, \mathcal{M}_o^{temp}, \mathcal{S}_n, \mathbb{T})$ *//Calculate the new lifetime quotient.*
13: $\quad\quad\quad\quad$ Compute $\mathcal{P}$ using Equation 3.24
14: $\quad\quad\quad\quad$ **if** $\mathbb{T}_n > \mathbb{T}_c$ and $\mathcal{P} > \mathcal{P}^{best}$ **then**
15: $\quad\quad\quad\quad\quad \mathcal{P}^{best} = \mathcal{P}$, $\mathcal{M}_d^{best} = \mathcal{M}_d^{temp}$, $\mathcal{M}_o^{best} = \mathcal{M}_o^{temp}$, $best\_found = true$, $\mathbb{T} = \mathbb{T}_n$
16: $\quad\quad\quad\quad$ **end if**
17: $\quad\quad\quad$ **end for**
18: $\quad\quad$ **end for**
19: $\quad$ **end for**
20: $\quad$ **if** $best\_found$ **then**
21: $\quad\quad \mathcal{M}_d = \mathcal{M}_d^{best}$ and $\mathcal{M}_o = \mathcal{M}_o^{best}$ *//Actor distribution and operating point matrices are updated.*
22: $\quad$ **else**
23: $\quad\quad$ break
24: $\quad$ **end if**
25: **end while**
26: Return $\begin{pmatrix}\mathcal{M}_d & \mathcal{M}_o\end{pmatrix}$ *//Actor distribution and operating point matrices are returned.*

3.5.1 Reliability Optimization for Individual Application

The objective function (lifetime quotient) of the optimization problem is non-linear; a gradient-based fast heuristic is proposed to solve it. This is shown as pseudo-code in Algorithm 4. The algorithm starts from an initial allocation, computed using the native SDF^3 tool (line 2). Subsequently, the algorithm remaps every actor to every core to determine a priority function defined as

$$\mathcal{P} = \begin{cases} \frac{lq_n - lq}{\mathbb{T} - \mathbb{T}_n} & \text{if } \mathbb{T}_n < \mathbb{T} \\ (lq_n - lq) & \text{otherwise} \end{cases} \tag{3.24}$$

Two conditions are considered in the priority computation: if the throughput of the current allocation ($\mathbb{T}_n$) is lower than the original throughput ($\mathbb{T}$), a gradient function is used to calculate its priority i.e.,assignments that increase the lifetime quotient with the least throughput degradation are given higher priorities. Conversely, if the current throughput is higher than the original one, high priorities are given to assignments with the largest increase in the lifetime quotient.

The algorithm remaps actor $\boldsymbol{a}_i$ to a core $\boldsymbol{c}_j$ at operating point k (lines 6 - 8). The actor

distribution and the operating point of actor $\boldsymbol{a}_i$ are changed (line 10). These matrices are used by the $MSDF^3$ tool to compute the throughput and schedule corresponding to the allocation $\mathcal{M}_d^{temp}$ (line 11). The $CalculateLQ$ function computes the lifetime quotient using Equation 3.11 to compute the energy and Algorithm 3 to compute the MTTF. The algorithm computes the priority function (line 13). If this priority is greater than the best priority obtained thus far and the throughput constraint is satisfied, the best values are updated (line 15). The algorithm continues to remap as long as an assignment can be found without violating the throughput requirement. When no such remapping is possible, the algorithm terminates.

3.5.2 Reliability Optimization for Use-cases

In this section, the use-case level optimization problem is formulated based on the results obtained in Section 3.5.1. It is to be noted that when multiple applications are enabled simultaneously, the temperature due to the execution of one application is dependent not only on the temperature of the cores on which it is executed, but also on the temperature due to other applications executing simultaneously. As a result, the wear-out (or the MTTF) due to single application can be significantly different than the actual wear-out (or the MTTF) for use-cases. This limitation is addressed using the pessimism introduced in the temperature model. Specifically, to determine the temperature for different cores during single application mode, all unused cores in the architecture (those, which can potentially execute other applications in multi-application use-case scenario) are considered to be operating, and their temperature effect are incorporated in determining the temperature of the actual operating cores. Although this gives a pessimistic bound on the temperature (and hence, the reliability), the approach simplifies the problem solution for multi-application use-cases.

As indicated previously, the *ThRiCoDB* contains 3D databases with throughput and MTTF number for every core count of every application. The problem addressed here is to merge these 3D databases for applications enabled simultaneously such that the distribution of the cores among these applications maximizes the system MTTF. For the

Algorithm 5 Core distribution for use-cases.

Input: $ThRiCoDB$
Output: Distribution of cores among applications
1: Initialize : $z_i = 0,\ 1 \leq i \leq n$
2: Initialize : $RiList.push(A_i, z_i, 0),\ 1 \leq i \leq n$
3: **for** $j = 1$ to N_c **do**
4: $\quad RiList.sort()$
5: $\quad$ Let, A_k = Task with least MTTF
6: $\quad z_k = z_k + 1$
7: $\quad M_k = ThRiCoDB.getMTTF(A_k, z_k)$
8: $\quad RiList.update(A_k, z_k, M_k)$
9: **end for**

ease of problem formulation, the following notations are defined:

$$
\begin{aligned}
A_1, \cdots, A_n &= \text{n applications enabled simultaneously} \\
z_i &= \text{number of cores for application } A_i \\
M_i &= \text{MTTF of } A_i \text{ mapped on } z_i \text{ cores} = ThRiCoDB.getMTTF(z_i) \\
\mathbb{T}_i &= \text{Throughput of } A_i \text{ mapped on } z_i \text{ cores} = ThRiCoDB.getThr(z_i)
\end{aligned}
$$

Formulation

The optimization problem is

$$
\begin{aligned}
\text{maximize} \quad & \min_i \{M_i\} \\
\text{subject to} \quad & \sum_{i=1}^{n} z_i \leq N_c \\
& \forall i, \mathbb{T}_i \geq \text{throughput constraint of } A_i
\end{aligned}
\tag{3.25}
$$

Solution

Algorithm 5 provides the pseudo-code to solve Equation 3.25. A list is defined ($RiList$) to store the applications (their IDs) of the use-case, the number of cores dedicated to it, and the corresponding MTTF value. For every core in the system (line 3), the $RiList$ is sorted to determine the application with the least MTTF (lines 4 - 5). A core is dedicated to this application (line 6); the corresponding MTTF is fetched from the $ThRiCoDB$ (line 7), and the $RiList$ is updated.

Table 3.1: ARM processor specification.

Power Mode	Frequency	Voltage (V)	Current (mA)	Power (mW)
OPP50	300MHz	0.93	151.62	141.01
OPP100	600MHz	1.10	328.79	361.67
OPP130	800MHz	1.26	490.61	618.17
OPP1G	1GHz	1.35	649.64	877.01

3.6 Experiments and Discussions

Experiments are conducted with real-life as well as synthetic SDFGs on a multiprocessor system with cores arranged in a mesh architecture. The synthetic SDFGs are generated using the SDF^3 tool [138] with the number of actors ranging from nine to twenty-five. These encompass both computation and communication dominated applications. The real-life SDFGs are *H.263 Encoder, H.263 Decoder, H.264 Encoder, MPEG4 Decoder, JPEG Decoder, MP3 Encoder and Sample Rate Converter*. Additionally, two non-streaming applications are also considered for this work. These are *FFT* and *Romberg Integration* from [142]. The supported voltage and frequency pairs are reported in Table 3.1, based on ARM Cortex-A8 core [143]. Although these voltage-frequency pairs are assumed for simplicity, the proposed algorithm and the temperature model can be trivially applied to any architecture with any supported voltage-frequency pairs.

The bit energy (E_{bit}) for modeling the communication energy of an application is calculated using the formulas provided in [41] for packet-based NoC with Batcher-Banyan switch fabric, using 65nm technology parameters from [144]. The parameters used for computing the MTTF are the same as in [95, 105, 106]. The scale parameter of each core is normalized so that its MTTF under idle (non-stressed) condition is 10 years. All algorithms are coded in C++, and used with SDF^3 tool for throughput and schedule construction, and *HotSpot* for temperature characterization. Additionally, Matlab regression toolbox is used for modeling the temporal and spatial dependency.

3.6.1 Time Complexity

The time complexity of the algorithms are calculated as follows. There are $(N_c - N_c^{min} + 1)$ loops in the algorithm 2 for each application. In each loop, the algorithm executes the $REOpt()$, the $MSDF^3()$, and the iterative technique to compute the MTTF (i.e., Algorithm 3). The complexity of Algorithm 3 is calculated as follows. Assuming lines 3

Table 3.2: Execution time (s) of the $MSDF^3$ tool with varying actors and cores.

Actors	Multiprocessor platform			
	6 cores	**9 cores**	**12 cores**	**16 cores**
8	3.1	7.6	7.6	7.6
16	6.8	10.1	26.8	101.1
24	217.4	241.7	323.0	409.8
32	899.4	1021.4	2211.0	2789.9

- 7 can be computed in unit time, the worst case complexity of this algorithm is

$$C_3 = O\left(N_c\right) \tag{3.26}$$

since $N_c^{min} \leq N_c$. The complexity of $REOpt()$ (Algorithm 4) is computed as follows. Let there be η iterations of the outer while loop (lines 3 - 25). In each iteration, the algorithm maps each actor to each core at each operating point to determine its reliability. The complexity of the this algorithm (C_4) is

$$C_4 = O\left(\eta \cdot N_a \cdot N_c \cdot N_f \cdot O(MSDF^3) \cdot C_3\right) \tag{3.27}$$

The $MSDF^3$ engine computes the schedule starting from a given actor distribution. This can be performed in $O\left(N_a \log N_a + N_a \cdot \pounds\right)$ (ref. [77]), where $\pounds$ is the average number of successors of an actor. Therefore,

$$C_4 = O\left(\eta \cdot N_a \cdot N_c \cdot N_f \cdot (N_a \log N_a + N_a \cdot \pounds) \cdot N_c\right) = O\left(N_a^5 \cdot N_f\right) \tag{3.28}$$

where $N_c, \pounds \leq N_a$. The overall complexity of the reliability-energy joint optimization for each application is $C_2 = O\left(C_4 + O\left(MSDF^3\right) + C_3\right) = O\left(N_a^5 \cdot N_f\right)$. The execution time of the $MSDF^3$ tool is reported in Table 3.2.

Finally, the complexity of Algorithm 5 is calculated as follows. For every iteration of the outer loop (number of cores), sorting of MTTF is performed once followed by the memory lookup. If the memory lookup time is assumed to be constant and there are n applications enabled simultaneously on N_c cores, every loop is executed in $O(n \log n)$. The overall complexity of Algorithm 5 is therefore $O(N_c \times n \log n)$. On the multiprocessor platform considered, this algorithm takes between 80-100μsec for two to six simultaneous applications on an architecture with nine homogeneous cores.

3.6.2 Validation of the Temperature Model

The temperature model in Equation 3.15 incorporates only the voltage and frequency of the one-hop neighbors with all other cores operating at the highest operating point of (1.35V, 1GHz). To determine the pessimism in this approach, Figure 3.8 plots the temperature variation obtained using the simplified model of Equation 3.15, in comparison

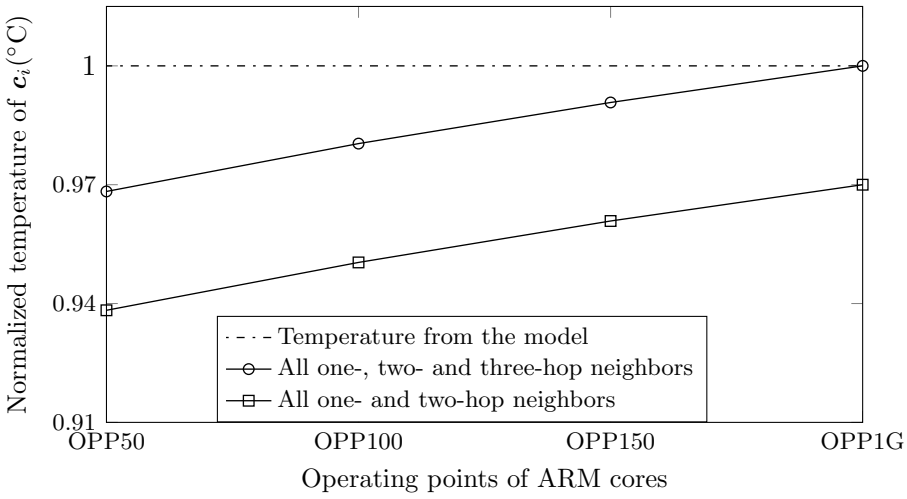


Figure 3.8: Temperature variation of the proposed model.

with the temperature obtained using the *HotSpot* tool by varying the operating points of the other neighbors. For this experiment, the execution time of the synthetic task is set to 300s to enable the proposed temperature model to reach its steady-state phase. The temperature data obtained from the *HotSpot* tool are the steady-state values generated by varying the operating point of core $\boldsymbol{c}_i$ and all of its one- and two-hop neighbors in lock-step, with all other cores set as idle. In terms of the *HotSpot* specification, this setup translates to varying the power of $\boldsymbol{c}_i$ and its one- and two-hop neighbors with the values from Table 3.1, and setting the power dissipation as zero for all other cores. The temperature of core $\boldsymbol{c}_i$ obtained from the *HotSpot* tool (in °C) is normalized with respect to the temperature obtained from the model for the different operating points. Similarly, the results for one-, two-, and three-hop neighbors are obtained. As seen from the figure, with the one- and two-hop neighboring cores operating at OPP50 (0.93V,300MHz), the temperature from the proposed model is an overestimate by 6.4% (9.5 °C in absolute terms). This overestimation decreases as the operating point is increased. This is expected, as more cores operate at the highest operating point, the temperature from the model is close to the temperature from the *HotSpot* tool. A similar trend is obtained for the one-, two-, and three-hop neighbors. For this plot, the temperature difference between the proposed model and the *HotSpot* tool is less than 0.1% at OPP1G.

3.6.3 Comparison with Accurate Temperature Model

Finally, the proposed temperature model is compared with the steady-state dynamic temperature profile (SSDTP) generated using the iterative technique of [97], and the

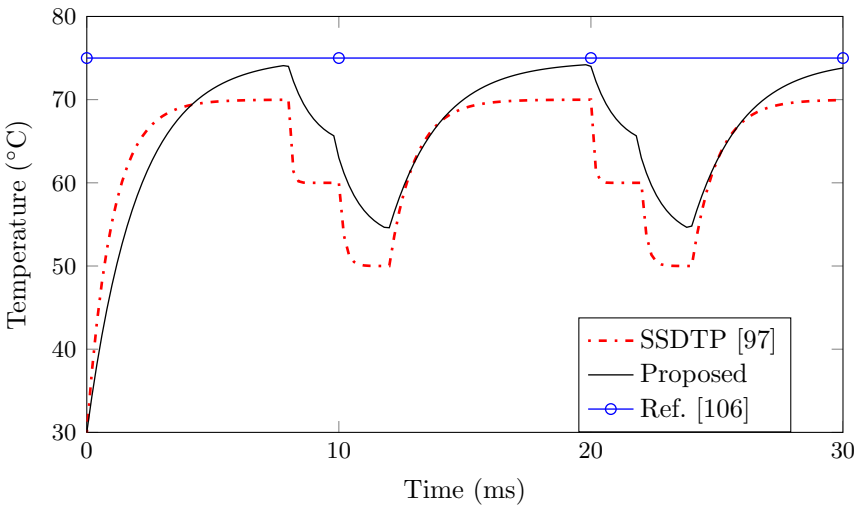


Figure 3.9: Comparison with accurate temperature model.

steady-state temperature model of [106]. A synthetic SDFG is considered for this experiment with a throughput requirement of 80 iterations per second. This translates to a steady-state period of 12.5ms. This SDFG is executed on a multiprocessor system with 9 cores. The steady-state iteration of the SDFG corresponds to a period of 12ms. The power profile of the SDGF varies within iteration, and this variable power profile is repeated every iteration. With such a variable power profile repeated periodically, the steady-state temperature is not constant, but varies according to the periodic power pattern as shown in Figure 3.9, with the red dashed line showing the results obtained using the temperature model of [97]. For the same power profile, the results with the proposed model are shown in the same figure with black solid line. The mean temperature for this two temperature plots are 63.5°C and 66.1°C, respectively. The temperature model of [106] assumes a steady-state value for the duration of operation, which corresponds to the average power in this duration. This is shown with blue solid line in the figure and corresponds to a temperature of 75°C. (11.5°C difference from the average temperature of [97]). Thus, in comparison to the temperature model of [97], the proposed temperature model is more accurate than the model of [106].

A point to note here is that, although the proposed model results in an average temperature close to that obtained using the accurate model of [97], the thermal cycling is not captured accurately leading to a misprediction of the thermal cycling related MTTF. However, the advantage is its simple form (non-iterative as opposed to the iterative technique of [97]), which can be included in the design space exploration, especially for multi-application use-cases.

Table 3.3: Impact of ignoring the temperature transient phase.

Apps	**MTTF using the model of [95]**	**MTTF using the model of [141]**	**MTTF using the *proposed model***
FFT	6.1	5.4	6.7
MPEG4	7.2	6.8	8.5
JPEG	8.6	9.4	9.6
MP3	6.4	6.1	7.5
SRC	7.9	8.7	8.7
synth16	6.8	6.0	6.8

3.6.4 Impact of Temperature Misprediction

As mentioned in Chapter 2, some existing techniques ignore the transient phase of the temperature. This leads to an inaccuracy in the temperature prediction and a corresponding inaccuracy in the MTTF computation. Furthermore, ignoring the spatial dependency leads to temperature misprediction. To establish the importance of the transient phase and the spatial dependency of the temperature on the MTTF results, an experiment is conducted with six applications (five real-life and one synthetic) on the multiprocessor platform with nine cores. Table 3.3 reports three MTTF values (in years): the MTTF obtained using the proposed technique with the temperature model of [95] that considers steady-state temperature phase only; the MTTF obtained using the proposed technique with the temperature model of [141] that considers the temporal dependency only; and the MTTF obtained using the proposed technique with the proposed temperature model.

For application FFT, the MTTF considering the proposed temperature model is 10% and 24% higher as compared to the MTTF considering the temperature model of [95] and [141], respectively. The MTTF improvement by ignoring the spatial dependency (column 3 vs column 4) is higher than the MTTF improvement ignoring the transient phase (column 2 vs column 4). A similar trend is observed for MP3 Decoder and H.264 Encoder. These results signify the importance of the spatial temperature component in the temperature estimation. A point to note here is that, the MTTF improvement by ignoring the spatial dependency is dependent on the size of the application executed on the platform. For JPEG application that uses only two cores of the architecture, the improvement is less than 3%. A similar trend is observed for Sample Rate Converter (SRC) application. Finally, as discussed in Section 3.1, considering the steady-state temperature is accurate only if the execution times of the actors of an application are comparable to the time constant of the RC equivalent circuit. This is shown for the synthetic application synth16 (with 16 actors) in the table. The execution times of the

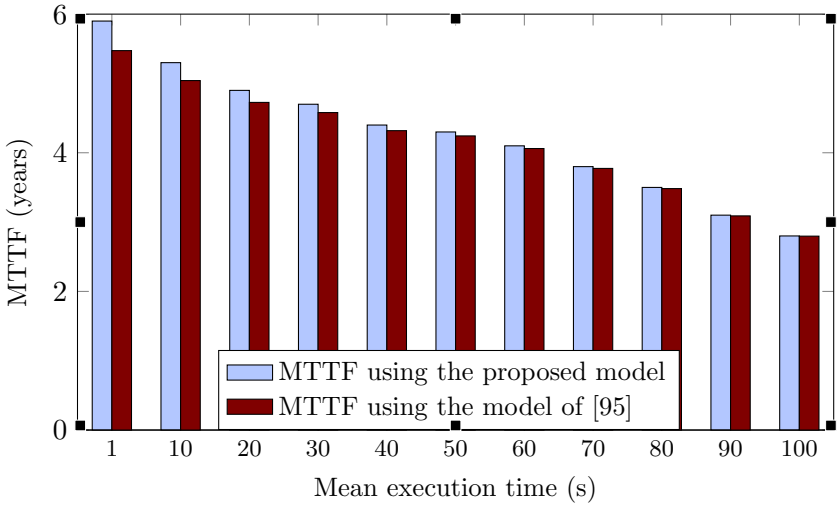
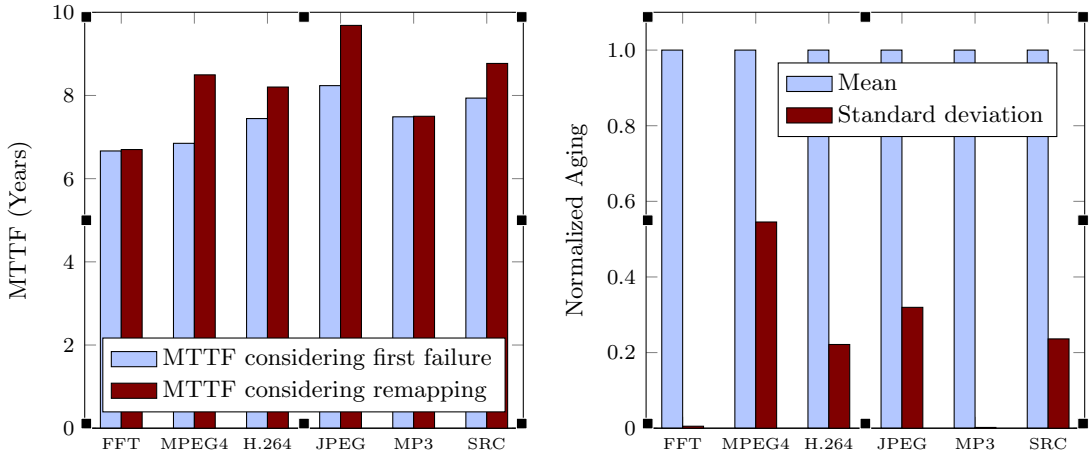


Figure 3.10: MTTF difference considering steady-state.

actors are generated with a mean of 200s and standard deviation of 20s. As can be seen, the MTTF obtained using the proposed model and the model of [95] are the same. On average for all the applications considered (seven real-life and synthetic), including the six shown in the table, the proposed model improves MTTF by 8% as compared to the model in [95], and 15% as compared to that of [141].

To give further insight into the temperature misprediction considering the steady-state temperature model of [95], experiments are conducted with an SDFG with 16 actors. The mapping and the schedule of this SDFG is generated using the SDF^3 tool. Next, the ordering of the actors on each core is retained (discarding the timing information), and the average execution times of the actors is varied from 1s to 100s in steps of 10s. The MTTF obtained using the temperature model of [95] is normalized with respect to the MTTF obtained using the proposed model. This is shown in Figure 3.10. The general trend to follow from this figure is that, the MTTF decreases with an increase in the average execution time. This is because, with increase in the average execution time of the actors, the stress on the system increases, reducing the MTTF. Furthermore, for small mean execution time, the MTTF using the proposed model is higher than that of [95] by 7%. As the mean execution time is increased, the two models become close and temperature difference is less than 0.01%.

Thus far, the validation of the proposed temperature model is presented. In the next few subsections, results to validate the proposed approach are presented.



(a) MTTF underestimation.

(b) Mean and standard deviation of aging.

Figure 3.11: MTTF considering task remapping.

3.6.5 MTTF Computation Considering Task Remapping

As indicated in Section 3.2.4, modern multiprocessor systems support remapping of tasks (actors in the SDFG terminology) on detection of faults. The MTTF for these systems need to be computed by considering task remapping, as opposed to the naive way of considering the time to the first fault. To determine the MTTF differences in the two computation techniques, experiments are conducted on a multiprocessor system with six cores and a set of six real-life applications. This is shown in Figure 3.11a. There are two bars for every application. The left bar is for the MTTF considering the first failure and the right one for MTTF considering re-mapping. As seen from the figure, the two MTTF values are similar for FFT and MP3 decoder applications. For the four other applications, the two MTTF values differ. On average for all applications considered, the MTTF improvement is 15%. To give more insight on the reason for such low MTTF difference for applications such as FFT, as opposed to say, JPEG decoder, Figure 3.11b plots the mean and the standard deviation of the aging of the different cores for the six applications. The standard deviation of the aging values is a measure of how much the aging of the individual cores differ from the mean value. A low standard deviation indicates a balanced situation with all the cores suffering similar wear-out. On the other hand, a high standard deviation indicates some cores age faster than others. The standard deviation is normalized with respect to the mean value of the aging.

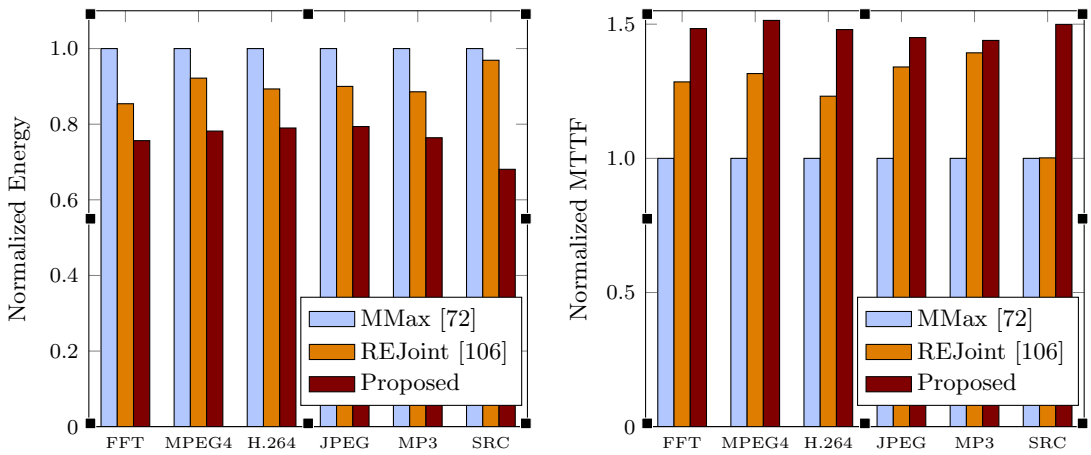
As seen from the figure, for applications such as FFT and MP3 Encoder, the standard deviation of the aging parameters is close to zero and thus the wear-out experienced in the

Table 3.4: *Processor years* considering task remapping.

Applications	PY considering TTFF	PY with task remapping			
		6 cores	5 cores	4 cores	Total
FFT	37.9	37.9	1.6	0.3	39.8
MPEG4	39.0	39.0	8.8	2.6	50.4
H.264	42.4	42.4	5.5	0.8	48.7
JPEG	46.9	46.9	8.2	2.4	57.5
MP3	42.6	42.6	1.6	0.3	44.5
SRC	45.2	45.2	6.0	0.9	52.1
Average					15.1%

cores due to these applications are similar. For these applications, the MTTF considering the first failure is similar (less than 0.5% lower on average) to the MTTF considering remapping. This is intuitive, because with all cores suffering similar wear-outs, the break point (the time at which a core fails due to wear-out) for all the cores are similar and therefore remapping leads to an insignificant gain in lifetime. For all the other applications, the standard deviations are high, with some applications having standard deviation as high as 60% of the corresponding mean value. For these applications, the aging values are not balanced. Although a balanced aging leads to a higher overall MTTF, a further investigation into these applications reveal that the balanced aging mapping for these applications consumes high energy; therefore, the proposed gradient-based heuristic selects the mapping with non-balanced aging, but with significantly low energy consumption. For these applications, the MTTF computation considering remapping is higher by as much as 24% (average 10%) than the MTTF computation considering the time to the first failure (TTFF).

Finally, Table 3.4 reports the *processor years* considering the time to the first failure and the overall lifetime considering task remapping. For demonstration purpose, only two faults are allowed, and therefore the table reports up to 4 cores used. Column 2 reports the *processor years* considering the time to the first failure. This is the aggregate years spent with all the 6 cores active. The *processor years* with task remapping are shown in columns 3, 4 and 5, with the total in column 6. Specifically, column 3 reports the aggregate years spent with all the 6 cores active; column 4 reports the aggregate years with 5 cores active; and column 5 reports the aggregate years spent with 4 active cores. As seen from the table, the total *processor years* considering task remapping for MPEG4 is 30% higher than the *processor years* considering TTFF. This improvement is due to the non-zero *processor years* with 5 and 4 active cores. This improvement signifies that,



(a) Energy comparison.

(b) MTTF comparison.

Figure 3.12: Energy-reliability joint optimization results.

even after the first fault, the multiprocessor system can be exploited to deliver 30% of the performance delivered during the time to the first fault. A similar trend is observed for the other applications in the table. On average, the *processor years* considering task remapping is 15% higher than the *processor years* considering the time to the first failure.

3.6.6 Reliability and Energy Improvement

Figure 3.12 plots the energy and reliability results of the proposed approach in comparison to the existing reliability-energy joint optimization technique of [106] for six real-life application. Additionally, to determine the reliability benefit of the dynamic voltage and frequency scaling, these two techniques are compared with the highest MTTF technique of [72] (referred to as MMax), which determines MTTF by solving a convex optimization problem. These results are represented as three bars corresponding to each application. A point to note here is that, all the application SDFGs are first converted to homogeneous SDFGs (HSDFGs) before applying the techniques of [72] and [106][3].

The following trends can be followed from the figure. The energy consumption using the proposed approach and the existing energy-reliability joint optimization technique of [106] are lower than the highest MTTF technique of [72] that does not consider dynamic voltage and frequency scaling. The MTTF obtained using these techniques are also higher than the MTTF of [72]. These results signify that, by slowdown of the actor computation, the reliability can be improved significantly.

[3]The conversion of an SDFG to HSDFG is of exponential complexity and therefore the proposed technique is the first technique for reliability-energy-performance optimization for SDFGs.

On average for all the applications considered, the existing optimization technique minimizes energy consumption by 10% with a corresponding reliability improvement of 26% as compared to the highest MTTF technique. A point to note here is that, this technique is based on sequential execution of applications; therefore, the throughput slack (difference between the actual throughput and the throughput constraint) is low, implying a limited scope for actor slowdown. The energy improvement in this technique is, therefore, not significant. The proposed technique achieves better results than this technique by minimizing energy consumption further by an average 15%, and increasing lifetime by an additional 18%. In comparison to [72], the proposed technique minimizes energy consumption by 24% and increases lifetime by 47%. These improvements can be attributed to

- the proposed temperature model that considers transient and steady-state phases as opposed to considering the steady-state temperature only;
- the MTTF computation considering remapping; and
- the pipelined scheduling technique of the proposed approach as opposed to the sequential execution of [106].

3.6.7 Design Space Exploration Speed-up

To highlight the speedup achieved by using the proposed design space exploration heuristic to jointly optimize energy and reliability, Table 3.5 reports its execution time in comparison with the convex optimization based technique of [72] and the simulated annealing based technique of [106]. The execution time are recorded by running synthetic SDFGs with varying number of actors on two multiprocessor systems – with four and six cores, respectively. The number of actors is limited to 8 as the convex optimization fails to provide results beyond 8 actors, even for running more than 12 hours. The time reported in this table are the average results obtained by generating multiple SDFGs. For example, the execution time for 6 actors on 4 cores is the average time taken by the three techniques for 10 different synthetic SDFGs, with 6 actors each. For fair comparison, the time taken by the temperature pre-characterization step is omitted for all these techniques and the time reported are the time for the respective technique – convex solver for [72], simulated annealing for [106], and proposed heuristic of Algorithm 4. As seen

Table 3.5: Design space exploration time with varying actors and cores.

Actors	Design Space Exploration Time (in minutes)					
	cores = 4			cores = 6		
	Convex [72]	**SA [106]**	***Proposed***	**Convex [72]**	**SA [106]**	***Proposed***
4	8	10	4	18	18	6
6	79	51	23	157	83	36
8	677	319	154	1013	524	274

from the table, for small number of actors the execution time of the convex solver is comparable to that of the simulated annealing (better for 4 actors on 4 cores). However, as the number of actors is increased, the simulated annealing outperforms the convex solver. In comparison to these two techniques, the proposed approach improves execution time significantly, achieving benefits for small and large problem sizes. On average, the execution time using the proposed technique is 70% and 50% lower with respect to [72] and [106], respectively.

3.6.8 Use-case Optimization Result

Since this work is the first work on use-case level MTTF optimization, there is no reference for comparison. However, two standard strategies are developed to distribute the cores among concurrent applications in a use-case – throughput-based core distribution (TCD) and equal core distribution (ECD). For implementing these approaches, the cores of the architecture are first distributed to the applications based on the corresponding strategy (equally or in the ratio of the throughput). The proposed optimization technique is then applied on individual applications to determine their MTTF. The overall MTTF of the use-case is the minimum of the MTTFs of the concurrent applications. The MTTFs obtained for a use-case using both these strategies, are compared with the MTTF obtained using the proposed MTTF-based core distribution technique. To demonstrate the advantage of the proposed approach for use-case optimization, a set of six synthetic use-cases are generated. Four of these uses-cases are composed of synthetic applications and the two others are composed of real-life applications. These use-cases are executed on a multiprocessor system with nine cores. Figure 3.13 plots the MTTF for the three approaches for these uses-cases. The composition of each use-case is indicated in the label of the figure, where the application with alphabets are the synthetic applications. For the use-case A-B, the MTTF obtained by distributing the cores equally is 4.6 years. The TCD achieves better results by distributing the cores in the ratio of their through-

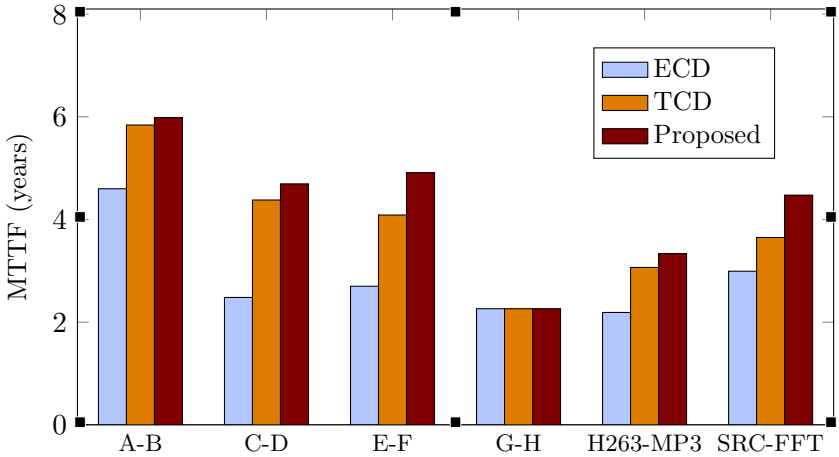


Figure 3.13: MTTF improvements with synthetic use-cases.

put requirements. The improvement in this technique is 27%. The proposed technique improves this further by achieving 3% higher lifetime. To understand the reason behind this improvement, a simple example is provided.

Let us consider a multiprocessor system with four cores and a use-case with two applications – synthA and synthB, with the throughput requirement of synthA as three time that of the synthB. The minimum number of cores required to satisfy the throughput requirement of synthA and synthB are two and one, respectively. Furthermore, let synthB stresses the system more (with a higher temperature) than synthA due to the higher execution time of the actors of synthB. Clearly, distributing the cores to these applications as 3:1 will not be optimal for MTTF. This example motivates and proves the importance of considering MTTF while distributing the cores of the architecture. As seen from the figure, for some use-cases, such as A-B and G-H, the improvements using the proposed technique are insignificant. For other use-cases, such as E-F and SRC-FFT, the improvements are more than 20%. On average for all these use-cases, the proposed technique improves MTTF by 10% as compared to TCD and 140% as compared to ECD.

3.7 Remarks

In this work, a simplified temperature model is proposed, based on off-line temperature characterization using the *HotSpot* tool. Based on this model, a gradient-based fast heuristic is proposed to determine the voltage and frequency of cores such that the energy consumption is minimized, simultaneously maximizing the system mean time to failure (MTTF). Experiments are conducted on a multiprocessor system using a set of synthetic

and real-life application SDFGs, executed individually as well as concurrently. Results demonstrate that the proposed approach minimizes energy consumption by an average 24% and maximizes lifetime by 47% as compared to the existing work. Additionally, the proposed MTTF-aware core distribution for concurrent applications results in an average 10% improvement in lifetime as compared to the performance-aware core distribution.

CHAPTER 4

Reliability and Energy-Aware Co-design Methodology

4.1 Introduction

As discussed in Chapter 1, an emerging trend in multiprocessor design is to integrate reconfigurable area alongside homogeneous processing cores. Hardware-software co-design of these reconfigurable multiprocessor systems needs to address the following two aspects:

Hardware-software task partitioning: Given a reconfigurable multiprocessor system and an application represented as a directed graph, the hardware-software task partitioning problem is to determine the tasks of the application that need to be executed on the processing cores (software tasks) and those required to be implemented as hardware on the reconfigurable area (hardware tasks). This problem has been studied extensively in literature to maximize performance and to minimize energy consumption [34].

Hardware sizing: The hardware sizing problem for reconfigurable multiprocessor systems is to determine the minimum resources (number of processing cores and the size of reconfigurable area) needed such that, the performance of every application (enabled individually or concurrently) is guaranteed while satisfying the design area budget [28].

Figure 4.1 shows the hardware-software co-design approach for the given set of applications. The approach invokes the hardware-software task partitioning step for every application iteratively, to check if the application performance is met. If this is violated,

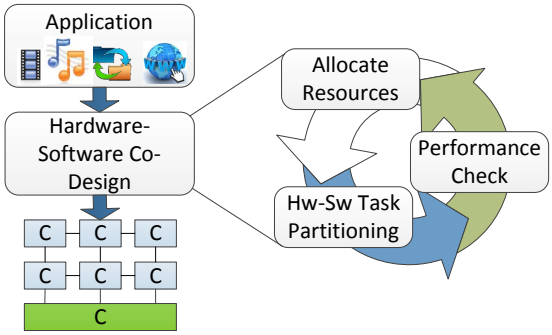


Figure 4.1: Hardware-software co-design methodology.

additional resources are allocated and the hardware-software task partitioning step is repeated; otherwise, the analysis terminates for the application and the process is invoked for the next application. The final platform is determined as the maximum of the resources of all the applications, enabled individually and concurrently.

The existing studies in this hardware-software co-design suffer from the following limitations. First, **checkpointing** is used for reconfigurable multiprocessor systems to tolerate transient faults in the processing cores. The solutions from these approaches guarantee or maximize fault-free task execution on the cores while exploiting the execution slack arising from the hardware execution of certain tasks. The area and performance overhead for the fault-tolerance of the logic implemented on the reconfigurable area are not accounted in these techniques. Further, the extent of fault-tolerance achieved within an allocated reconfigurable area in a co-design framework is not addressed. A complete solution to fault-tolerance needs to incorporate the transient fault-tolerance overhead for both the software and hardware tasks while satisfying the design performance constraints and reconfigurable area availability.

Second, none of the existing hardware-software task-partitioning techniques consider wear-out of the processing cores and transient faults simultaneously. As shown in Section 4.2.4, improving the transient fault-tolerance by increasing the number of checkpoints, negatively impacts lifetime reliability of the cores due to wear-out. A balance of the two is essential to mitigate transient faults and wear-out jointly for multiprocessor systems. Moreover, the only existing wear-out aware co-design technique for a static multiprocessor system leaves a significant scope of improvement both in terms of reliability and resource usage when applied to a reconfigurable system.

Third, multimedia applications such as H.264 encoder, decoder, JPEG decoder, etc., are characterized by cyclic dependency of tasks, and require a fixed throughput to be

satisfied to guarantee quality-of-service to end users. Existing studies on hardware-software co-design are based on acyclic graph model of applications with no consideration of throughput degradation. These techniques require significant modification (if at all applicable) for streaming multimedia applications represented as synchronous data flow graphs (SDFGs). Last, none of the existing works consider multi-application use-cases, which is a common requirement for most multiprocessor systems.

In this work, a design-time technique is proposed for hardware-software partitioning of an application i.e., determining the tasks to be executed on the processing cores and those on the reconfigurable area. The objective is to improve fault-tolerance of the platform considering three effects – transient faults occurring in the processing cores, single event upsets occurring in the logic configuration bits of the reconfigurable area, and the wear-out of the processing cores. The transient faults in the cores are mitigated using checkpoints[1]. Single event upsets in the logic configuration bits of the reconfigurable area (like Xilinx FPGA) manifest as permanent faults and render the affected logic useless, unless reprogrammed. The proposed approach does not consider reprogramming the reconfigurable area within an application execution; therefore, redundancy-based techniques are used for the single event upsets. The corresponding area overhead is incorporated in the problem formulation. Finally, wear-out of the cores is mitigated using intelligent task mapping and scheduling. Based on the proposed hardware-software task partitioning technique, a hardware-software co-design approach is proposed to determine the minimum resources needed to map and guarantee throughput of applications in all use-cases, simultaneously improving the lifetime reliability measured as mean time to failure (MTTF) and satisfying the specified energy budget and design cost. Following are the key contributions.

- Formulation of the wear-out of processing cores and checkpoint-based transient error recovery problem in the hardware-software task partitioning framework with reconfigurable area as a constraint;
- Design space exploration for application partitioning for reconfigurable multiprocessor systems;

[1]Of the different transient fault-tolerant techniques available for processing cores, such as error correction coding, duplication with re-execution and checkpointing as highlighted in Chapter 1, we selected checkpointing as this provides the best trade-off between fault-tolerance and execution/resource overhead.

- Reliability-aware hardware-software co-design framework incorporating the proposed design space exploration technique;
- Integer linear programming (ILP)-based merging of Pareto-optimal solutions for individual applications to determine the resource requirement for multi-application use-cases;
- Considering SDFG for the hardware-software co-design of multiprocessor system.

The remainder of this chapter is organized as follows. The proposed reliability-aware hardware-software task partitioning technique is discussed in Section 4.2. The co-design framework is introduced next in Section 4.3. Results are presented in Section 4.4 and the chapter is concluded in Section 4.5.

4.2 Reliability-Aware Hardware-Software Task Partitioning

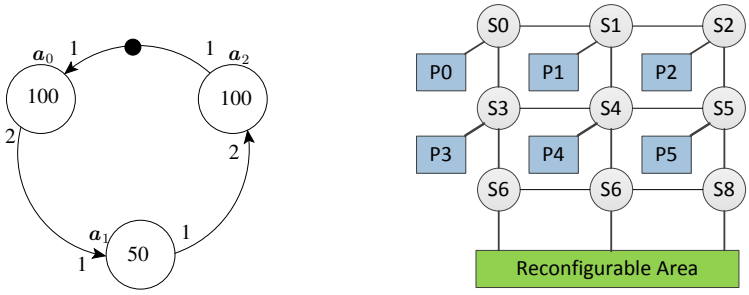
In this section, an application partitioning technique is introduced for reconfigurable multiprocessor systems with reliability as the optimization objective. The proposed technique generates the following decisions for every application enabled on the platform.

1. Tasks to be mapped on the processing cores (software tasks) and tasks to be implemented on the reconfigurable area (hardware tasks);
2. Number of checkpoints for each software tasks; and
3. Mapping and scheduling of the software and hardware tasks on the given platform.

These decisions are computed offline at design-time; the mapping and the reconfigurable area configuration (e.g., bit stream) are stored in a database for every application. When an application is enabled at run-time, the corresponding mapping and configuration data ate fetched. The reconfigurable area is programmed with the configuration data, and finally the mapping is applied to execute the tasks of the application. The proposed approach does not incorporate dynamic partial reconfiguration i.e., reconfiguring the fabric during an application execution. This is left as a future work.

4.2.1 Application and Architecture Model

An application is represented as an SDFG $\mathcal{G}_{app} = (\mathbb{A}, \mathcal{C})$, where $\mathbb{A}$ is the set of actors representing tasks of the application and $\mathcal{C}$ is the set of directed edges representing



(a) Applications model (b) Architecture model

Figure 4.2: Application and architecture model.

data dependency among various actors. The number of actors is represented as N_a i.e., ($N_a = |\mathbb{A}|$). Every actor $\boldsymbol{a}_i \in \mathbb{A}$ is a tuple $\langle RA_i, t_i, \tau_i \rangle$, where RA_i is the area required to implement $\boldsymbol{a}_i$ on the reconfigurable area, t_i is the time taken by $\boldsymbol{a}_i$ to execute on the dedicated hardware, and τ_i is its execution time on the core. If an actor does not support hardware implementation, the value of this parameter is set to infinite. The area overhead for redundancy-based transient fault-tolerance for actor $\boldsymbol{a}_i$ is incorporated into RA_i. For actors requiring fault-detection only, RA_i is the area of duplicating the logic and the area of a checker circuit. For those actors requiring fault-mitigation, RA_i includes the area for triple-modular redundancy (TMR) and the voter circuit.

The architecture (refer to Figure 4.2b) consists of N_c homogeneous cores connected to a shared reconfigurable area. The reconfigurable area is a one-dimensional (1D) model and is divided into N_r equal sized frames. A frame is a basic unit for reconfiguration (e.g., Xilinx Virtex 6 FPGA). The architecture is represented as $\mathcal{G}_{arc} = (\mathbb{C}, \mathbb{E})$, where $\mathbb{C}$ is the set of cores and $\mathbb{E}$ is the set of links connecting the cores. The number of cores in the architecture is denoted by N_c i.e., $N_c = |\mathbb{C}|$. For the ease of problem formulation, the reconfigurable area is considered as a virtual core and the modified graph is denoted as $\mathcal{G}_{arc}^{N_c+1}$. The following restrictions and relaxations apply for the virtual core.

1. Actors mapped to the virtual core have associated area cost. This is because, an actor mapped on the virtual core implies dedicated hardware for the actor (in reality) that consumes few frames of the reconfigurable area.

2. Multiple independent actors can be executed at the same time on the virtual core. This is because, independent actors implemented on different regions of the reconfigurable area can run independently and concurrently. Although this leads to a contention on the communication link between the processing cores and the

reconfigurable area, investigation on the communication aspect of reconfigurable multiprocessor systems is left as future work.

3. An actor mapped on the virtual core does not need software-based protection techniques, such as **checkpointing** and **rollback**. Instead, the protection is provided by replicating the hardware implementation.

This work analyzes the trade-off between transient fault-tolerance and lifetime reliability, considering the impact of temperature resulting from different mapping and scheduling alternatives (refer to Chapter3 for computing temperature for a schedule). In future, the work can be extended to consider dynamic voltage and frequency scaling.

The mapping is specified in terms of the *actor distribution* variable $x_{i,j}$ and the *checkpoint assignment* variable $z_{i,k}$ defined as

$$x_{i,j} = \begin{cases} 1 & \text{if actor } \boldsymbol{a}_i \text{ is assigned to core } \boldsymbol{c}_j \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

$$z_{i,k} = \begin{cases} 1 & \text{if actor } \boldsymbol{a}_i \text{ is assigned } k \text{ checkpoints} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

The mapping of $\mathcal{G}_{app}$ on $\mathcal{G}_{arc}$ is represented in terms of two matrices $\begin{pmatrix} \mathcal{M}_d & \mathcal{M}_c \end{pmatrix}$, where $\mathcal{M}_d$ and $\mathcal{M}_c$ are defined as

$$\mathcal{M}_d = \begin{pmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,N_c} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,N_c} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N_a-1,0} & x_{N_a-1,1} & \cdots & x_{N_a-1,N_c} \end{pmatrix} \tag{4.3}$$

$$\mathcal{M}_c = \begin{pmatrix} z_{0,0} & z_{0,1} & \cdots & z_{0,N_c-1} \\ z_{1,0} & z_{1,1} & \cdots & z_{1,N_c-1} \\ \vdots & \vdots & \ddots & \vdots \\ z_{N_a-1,0} & z_{N_a-1,1} & \cdots & z_{N_a-1,N_c-1} \end{pmatrix} \tag{4.4}$$

It is to be noted that the actor distribution matrix $\mathcal{M}_d$ includes the mapping variable for the virtual core (i.e., the RA). Throughout the rest of this chapter, the virtual core is indexed by N_c and the homogeneous cores using indices $0, 1, \cdots, N_c - 1$.

4.2.2 Reliability Modeling Considering Single Actor

Checkpoint refers to the state of the system at a particular instance of time. The process of **checkpointing** involves periodic storing of the checkpoints (in local or remote

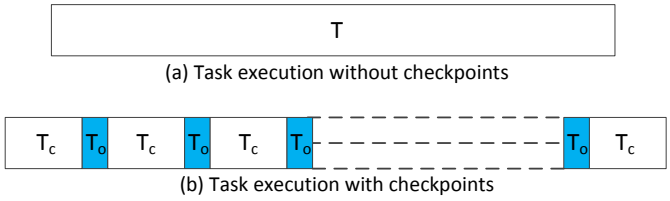


Figure 4.3: Task execution with and without checkpoints.

memory) by suspending an actor execution. The interval between two successive checkpoints is called the **checkpoint interval**. The actor resumes execution at the beginning of each checkpoint interval. When transient faults occur during a checkpoint interval, the useful computation of that interval is discarded and the execution of the actor is repeated from the last valid checkpoint. A point to be noted is that, the discussions in this work are limited to transient fault-tolerance and is orthogonal to any fault detection techniques such as [145]. One important parameter of checkpointing is the **checkpoint overhead**, defined as the increase in the execution time. This overhead depends on

1. the number of checkpoints, N;
2. the time for checkpoint capture and storage, T_o;
3. the time for recovery from a checkpoint, T_r; and
4. the fault arrival rate, λ.

Following are the assumptions for checkpointing, similar to the works in [119, 146].

- transient faults follow Poisson distribution with a rate of λ failures per unit time;
- transient faults are point failures i.e., these faults induce errors in the checkpoint interval once and then disappear;
- these faults are statistically independent; and
- checkpoints can be inserted anywhere during an actor execution. Although this assumption is difficult to accomplish in practice, it gives a first-order approximation of the problem at hand.

Figure 4.3 shows an example actor execution with N checkpoints. Let T denote the actual computation time of the actor and T_c, the computation time of the actor in each checkpoint interval. Clearly, $T_c = \frac{T}{N+1}$. Assuming Poisson fault arrival, the probability

of k faults in the interval t and $t + \Delta t$ is

$$P(t, t + \Delta t, k) = \frac{e^{-\lambda \Delta t} (\lambda \Delta t)^k}{k!} \text{ for } k = 0, 1, \cdots \tag{4.5}$$

Therefore, the probability of at least one fault in the interval Δt is

$$P(t, t + \Delta t, k \geq 1) = 1 - P(t, t + \Delta t, k = 0) = 1 - e^{-\lambda \Delta t} \tag{4.6}$$

Assuming the interval Δt as the checkpoint interval $(T_c + T_o)$, the probability of at least one fault in this interval is $P_e = 1 - e^{-\lambda(T_c+T_o)}$. The expected length of checkpoint interval $E[T_c]$ is calculated as

$$E[T_c] = P\{\text{no fault}\} \cdot \text{ original checkpoint interval } + P\{\text{fault}\} \cdot \text{modified checkpoint interval} \tag{4.7}$$

When there are no faults in a checkpoint interval, the duration of this interval is $T_c + T_o$, where T_o is the time for checkpoint computation and storage (refer to Figure 4.3). Let T_f denote the time of the first fault from the start of a checkpoint interval. Since faults can occur at any time in the checkpoint interval with a probability P_e, T_f is uniformly distributed in the range 0 to $(T_c + T_o)$ with an average value of $\frac{T_c+T_o}{2}$. Hence, the modified checkpoint interval is given by $T_f + T_r + (T_c + T_o)$, where the first term is the useful computation lost since the beginning of the checkpoint interval, the second term is the time for recovery from the last valid checkpoint, and the last term is the re-execution time of the checkpoint segment starting from the last valid checkpoint. The recovery time includes the overhead for fetching the checkpoint from the local or remote memory and re-loading the registers of the core. Thus, Equation 4.7 can be written as

$$E[T_c] = (1 - P_e) \cdot (T_c + T_o) + P_e \cdot (\tau + T_r + T_c + T_o) = \frac{3(T_c + T_o)}{2} + T_r - \left(\frac{T_c + T_o + 2T_r}{2}\right) e^{-\lambda(T_c+T_o)} \tag{4.8}$$

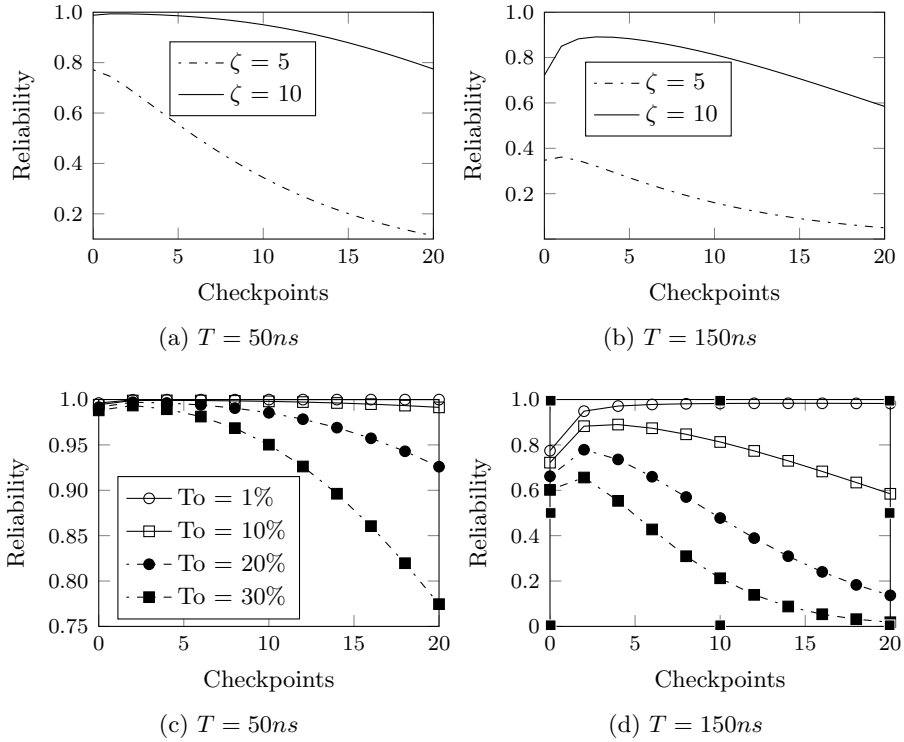
The expected length of the last checkpoint interval $(E[T_c^L])$ is computed from the above Equation by replacing $(T_c + T_o)$ with T_c. This is because there is no checkpoint overhead for the last interval. The expected execution time of the actor is given by

$$E[T] = N \cdot E[T_c] + E[T_c^L] \tag{4.9}$$

The reliability of an actor $\boldsymbol{a}_i$ with N checkpoints is given by

$$R_i^C(t) = (1 - P_e)^{N+1} + \binom{N+1}{1} P_e (1 - P_e)^{N+1} + \binom{N+2}{2} P_e^2 (1 - P_e)^{N+1} + \cdots \tag{4.10}$$

where the first term on the right hand side is the reliability with no faults, the second term is the reliability with one transient fault in any of the $N+1$ checkpoint interval, the third term is the reliability with two faults in $N+2$ intervals ($N+1$ original intervals and



(a) $T = 50ns$ (b) $T = 150ns$

(c) $T = 50ns$ (d) $T = 150ns$

Figure 4.4: Impact of different parameters on the reliability considering transient faults.

1 re-execution interval of the interval where the first fault occurs), and so on. Assuming infinite faults in the task execution, the above expression reduces to

$$R_i^C(t) = \sum_{\omega=0}^{\infty} \binom{N+\omega}{\omega} P_e^{\omega}(1-P_e)^{N+1} = 1 \tag{4.11}$$

This result is intuitive because if re-execution is allowed every time a fault is detected, the actor will eventually be executed successfully. However, for real time systems with throughput constraints, infinite faults will lead to throughput violation. Therefore, for real time systems, the sum in Equation 4.11 is evaluated from 0 to ζ_i, where ζ_i is the maximum number of faults that can be tolerated for actor $\boldsymbol{a}_i$ such that its throughput constraint is satisfied. The reliability of actor $\boldsymbol{a}_i$ is

$$R_i^C(t) = \sum_{\omega=0}^{\zeta_i} \binom{N+\omega}{\omega} P_e^{\omega}(1-P_e)^{N+1} \tag{4.12}$$

Figures 4.4a and 4.4b plot this reliability obtained for execution time $T = 50ns$ and $T = 150ns$, respectively as the number of checkpoints is increased from 0 (implying no checkpoints) to 20. The reported results are for ζ values of 5 and 10. As seen from the figure, the reliability first increases with the number of checkpoints up to a certain number (called the reliability drop-off point) beyond which, the reliability starts decreasing. This trend is consistent to that discussed in [147, 148]. Moreover, the reliability increases with an increase in the number of faults that can be tolerated i.e. ζ (consistent

with Equation 4.12). Finally, the reliability, corresponding to a particular number of checkpoints and ζ value, decreases as the actor execution time is increased from 50ns to 150ns. This is because, with increase in the execution time, the length of the checkpoint interval increases ($T_c = \frac{T}{N+1}$) and so does the fault probability P_e (Equation 4.6). This reduces the reliability.

Figures 4.4c and 4.4d plot the dependency of the checkpoint overhead (T_o) on the highest reliability value. The checkpoint overhead is expressed as a percentage of the execution time in these figures. As seen from these figures, the highest reliability point decreases with an increase in the overhead. This is expected because, with increase in the checkpoint overhead, the time between two successive checkpoints ($T_c + T_o$) increases, leading to an increase of the fault probability.

4.2.3 Reliability Modeling Considering Multiple Interconnected Actors

Lifetime Reliability: The MTTF of a system considering an application with multiple interconnected actors is given by the *Max-Min* approach (Chapter 2). Thus,

$$MTTF = \min_j \{MTTF_j\} \tag{4.13}$$

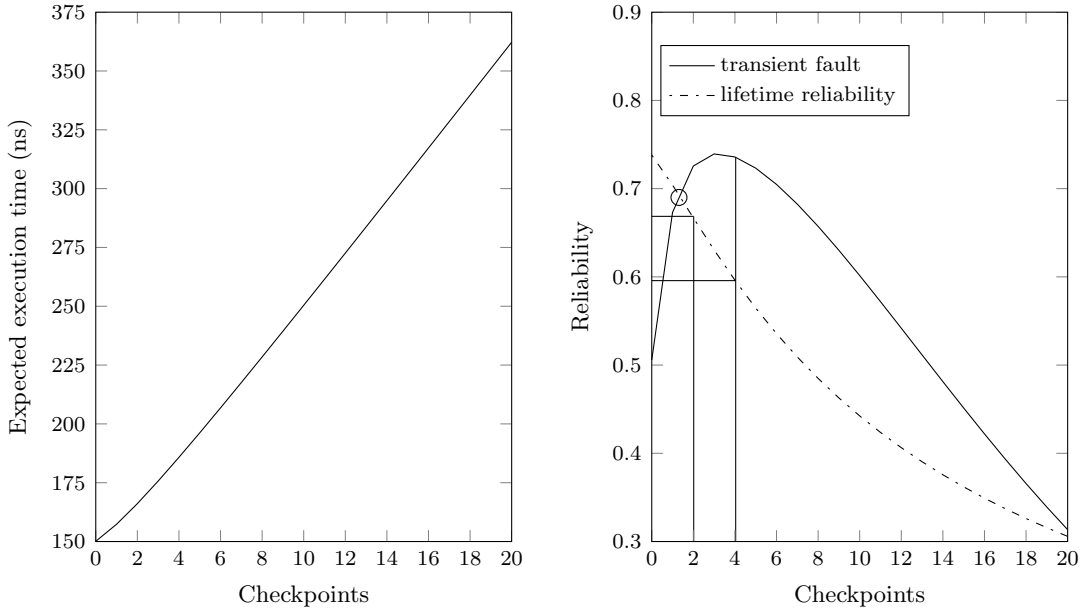
Reliability Considering Checkpoints: For an application consisting of multiple interconnected actors, the overall reliability considering transient faults is derived based on the assumption that transient fault occurrences are independent of each other, and an application is successful when all the actors of an application execute successfully. The reliability and the mean time between failures (MTBF) are given by

$$R_T(t) = \prod_{i=1}^{N_a} R_i^C(t) \text{ and } MTBF = \int_{t=0}^{\infty} R_T(t)dt \tag{4.14}$$

where N_a is the number of actors and $R_i^C(t)$ is the reliability of actor $\boldsymbol{a}_i$.

4.2.4 Lifetime Reliability and Transient Fault Reliability Trade-off

Figure 4.5a plots the expected execution time of an actor as the number of checkpoints is increased. The parameters used for simulation are as follows: execution time with no-checkpoints, $T = 150nS$, the checkpoint computation and storage overhead $T_o = 15nS$, the recovery time $T_r = 0.5nS$, and a transient fault rate of one fault every 100 hours. As seen from the figure, the expected execution time increases with an increase in the number of checkpoints. With every extra checkpoint, the checkpoint interval reduces



(a) Expected execution time

(b) Reliability trade-off

Figure 4.5: Trade-off between lifetime reliability and transient faults related reliability.

and therefore, the loss in computation reduces when a fault is detected. However, with every added checkpoint, the corresponding checkpoint overhead is to be included in the execution time. This increases the expected execution time (Equation 4.9).

As shown in Chapter 2, the lifetime reliability of a core due to wear-out is negatively dependent on the execution time of the actors executed on the core. Figure 4.5b plots the decrease in lifetime reliability (considering wear-out) with increase in the number of the checkpoints. This is according to Equation 2.12, and is shown in the figure with the dotted line. To highlight the trade-off between lifetime reliability and the reliability considering transient faults, Equation 4.12 is also plotted on the same figure, and is shown with the solid line.

As seen from the figure, the two reliability plots cross-over at a certain number of checkpoints. This cross-over point is dependent on the execution time, the number of checkpoints, and the overheads T_o and T_r. Since the transient fault reliability first increases and then decreases, there can be potentially two cross-over points. Since the number of checkpoints needs to be an integer, the selected values are rounded to the nearest integer. As an example, the cross-over point corresponds to 1.8 checkpoints, which is rounded to 2. Similarly, the number of checkpoints corresponding to the highest transient fault reliability is 3.6 and is rounded to 4. There are two lines drawn in the figure corresponding to 2 and 4 checkpoints. As seen from this figure, selecting 4 checkpoints

Algorithm 6 $HSAP()$: Hardware-software actor partitioning

Input: $\mathcal{G}_{app}$, $\mathcal{G}_{arc}^{N_c+1}$, throughput constraint $\mathbb{T}_c$, size of reconfigurable area S_{RA}
Output: $\begin{pmatrix}\mathcal{M}_d & \mathcal{M}_c\end{pmatrix}$
1: Initialize $HList = \emptyset$
2: **while** $\sum_{\forall \boldsymbol{a}_k \in HList} RA_k \leq S_{RA}$ **do**
3: **for all** $\boldsymbol{a}_i \in \mathbb{A} \setminus HList$ **do**
4: $HList.push(\boldsymbol{a}_i)$
5: $[\mathcal{M}_d \;\; \mathcal{M}_c \;\; rg^i] = FindMinRG(\mathcal{G}_{app}, \mathcal{G}_{arc}^{N_c+1}, \mathbb{T}_c, HList)$
6: $HList.pop(\boldsymbol{a}_i)$
7: **end for**
8: Find $\boldsymbol{a}_j \in \mathcal{G}_{app} \setminus HList$ such that rg^j is minimum
9: $HList.push(\boldsymbol{a}_j)$
10: **end while**
11: $[\mathcal{M}_d \;\; \mathcal{M}_c \;\; rg] = FindMinRG(\mathcal{G}_{app}, \mathcal{G}_{arc}^{N_c+1}, \mathbb{T}_c, HList)$
12: Return $\begin{pmatrix}\mathcal{M}_d & \mathcal{M}_c\end{pmatrix}$

(corresponding to the highest transient fault reliability) results in a reduction of the lifetime reliability by 15% as compared to selecting 2 checkpoints (corresponding to the cross-over point). Clearly a trade-off analysis needs to be performed to select the number of checkpoints for every actor of a given application.

4.2.5 Hardware-Software Partitioning Flow

To simplify the objective function of the hardware-software actor partitioning, a joint metric **reliability gradient** (rg) is defined as

$$rg = \frac{\Delta R_P(t)}{\Delta R_T(t)} \tag{4.15}$$

where $R_T(t)$ and $R_P(t)$ are the reliability considering transient faults and wear-out, respectively. The reliability gradient is interpreted as ratio of the change (decrease) in reliability due to wear-out per unit change (increase) in reliability due to transient faults. The optimization objective is to minimize the reliability gradient. A fast design space exploration technique is proposed. This is shown as pseudo-code in Algorithm 6.

A list ($HList$) is defined to store the hardware actors i.e., actors that are to be implemented on the reconfigurable area. The algorithm iterates (lines 2 - 10) as long as the available reconfigurable area constraint is satisfied (line 2). At every iteration, the algorithm selects one actor from the set $\mathbb{A} \setminus HList$ (i.e. selects one of those actors not marked as hardware actors) and assigns it temporarily to the reconfigurable area (line 4). The hardware actors (from $HList$) are mapped on the reconfigurable area and the software actors on the processing cores, and the whole application graph is scheduled to determine the minimum reliability gradient. This step is performed using the $FindMinRG$ routine that outputs the actor distribution, checkpoint assignment and

Algorithm 7 $FindMinRG()$: Mapping and scheduling to find the minimum reliability gradient

Input: $\mathcal{G}_{app}$, $\mathcal{G}_{arc}^{N_c+1}$, throughput constraint $\mathbb{T}_c$ and $HList$
Output: Mapping $\begin{pmatrix}\mathcal{M}_d & \mathcal{M}_c\end{pmatrix}$ and minimum reliability gradient rg

1: $[\mathcal{M}_d \;\; \mathcal{S}] = REOpt(\mathcal{G}_{app}, \mathcal{G}_{arc}^{N_c+1}, \mathbb{T}_c, 1)$ *//Wear-out minimum mapping.*
2: Initialize $\mathcal{M}_c$ with $z_{i,k} = 0 \; \forall i, k$ *//Set checkpoints for all actors equal to zero.*
3: $[R_P^{init} \;\; R_T^{init}] = CalculateReliability(\mathcal{S}, \mathcal{M}_d, \mathcal{M}_c)$ *//Calculate the initial wear-out and transient fault related reliability using Equations 2.12 and 4.14.*
4: Initialize $runIter = 1$ *//Initialize the loop variable.*
5: **while** $runIter > 0$ **do**
6: $\quad [\mathcal{S} \;\; \mathbb{T}] = MSDF3(\mathcal{M}_d, \mathcal{G}_{app}, \mathcal{G}_{arc}^{N_c+1})$ *//Get the schedule and throughput of the allocation matrix $\mathcal{M}_d$.*
7: $\quad [R_P \;\; R_T] = CalcReliability(\mathcal{S}, \mathcal{M}_d, \mathcal{M}_c)$ *//Calculate reliabilities.*
8: $\quad i_{best} = -1$; $j_{best} = -1$; $k_{best} = -1$; $rg_{best} = \infty$ *//Initialize.*
9: $\quad$ **for all** $a_i \in \mathbb{A} \setminus HList$ **do**
10: $\quad\quad$ **for all** $c_j \in \mathcal{G}_{arc}^{N_c+1}$ **do**
11: $\quad\quad\quad$ **for** $k = 1$ to N_{cp} **do**
12: $\quad\quad\quad\quad \begin{pmatrix}\mathcal{M}_d^{temp} & \mathcal{M}_c^{temp}\end{pmatrix} = \begin{pmatrix}\mathcal{M}_d & \mathcal{M}_c\end{pmatrix}$ *//Use temporary allocation and assignment matrices.*
13: $\quad\quad\quad\quad$ Update $\begin{pmatrix}\mathcal{M}_d^{temp} & \mathcal{M}_c^{temp}\end{pmatrix}$ with $x_{i,j} = z_{i,k} = 1$ *//Update these matrices.*
14: $\quad\quad\quad\quad [\mathcal{S} \;\; \mathbb{T}] = MSDF3(\mathcal{M}_d^{temp}, \mathcal{G}_{app}, \mathcal{G}_{app}^{N_c+1})$ *//Calculate the new schedule and throughput.*
15: $\quad\quad\quad\quad [R_P^{temp} \;\; R_T^{temp}] = CalcReliability(\mathcal{S}, \mathcal{M}_d^{temp}, \mathcal{M}_c^{temp})$
16: $\quad\quad\quad\quad rg = \frac{R_P - R_P^{temp}}{R_T^{temp} - R_T}$ *//Calculate the reliability gradient.*
17: $\quad\quad\quad\quad$ **if** $rg < rg_{best}$ && $\mathbb{T} \geq \mathbb{T}_c$ **then**
18: $\quad\quad\quad\quad\quad i_{best} = i$; $j_{best} = j$; $k_{best} = k$; $rg_{best} = rg$
19: $\quad\quad\quad\quad$ **end if**
20: $\quad\quad\quad$ **end for**
21: $\quad\quad$ **end for**
22: $\quad$ **end for**
23: $\quad$ **if** $i_{best} \geq 0$ **then**
24: $\quad\quad$ Update $\begin{pmatrix}\mathcal{M}_d & \mathcal{M}_c\end{pmatrix}$ with $x_{i_{best}, j_{best}} = z_{i_{best}, k_{best}} = 1$
25: $\quad$ **else**
26: $\quad\quad runIter = 0$ *//No possible remapping w/o violating the throughput constraint.*
27: $\quad$ **end if**
28: **end while**
29: $[\mathcal{S} \;\; \mathbb{T}] = MSDF3(\mathcal{M}_d, \mathcal{G}_{app}, \mathcal{G}_{app}^{N_c+1})$ *//Schedule and throughput of the final allocation.*
30: $[R_P^{finl} \;\; R_T^{finl}] = CalculateReliability(\mathcal{S}, \mathcal{M}_d, \mathcal{M}_c)$ *//Final reliabilities.*
31: $rg = \frac{R_P^{init} - R_P^{finl}}{R_T^{finl} - R_T^{init}}$ *//Overall reliability gradient.*
32: Return $[\mathcal{M}_d \;\; \mathcal{M}_c \;\; rg]$

the reliability gradient (line 5). The actor, whose assignment to the reconfigurable area leads to the minimum reliability gradient, is permanently marked as hardware actor and pushed to the $HList$ (line 9). The key component of this algorithm is the $FindMinRG$ subroutine, whose pseudo-code is shown in Algorithm 7.

The first step of Algorithm 7 is the initial actor distribution that minimizes wear-out (line 1). The algorithm continues to remap ever actor to a core with different checkpoints to determine the reliability gradient (lines 9 - 16). If the reliability gradient obtained for an assignment is lower than the best value obtained thus far, the best values are updated (lines 17 - 19). After iterating for all the actors, the actor distribution is changed with the best value of actor, core and checkpoints. This process is re-iterated (starting with this changed distribution) as long as no further re-mapping is possible without violat-

ing the throughput constraint. When this happens, the best value of actor, core and checkpoints are all negative. The algorithm proceeds to the **else** section (lines 25 - 27) where the terminating condition is asserted. The final actor distribution and checkpoint assignment matrices $\begin{pmatrix} \mathcal{M}_d & \mathcal{M}_c \end{pmatrix}$ are returned along with the overall reliability gradient. The actor distribution matrix is used in the $MSDF^3$ tool (refer to Chapter 3) to generate the throughput and schedule. The schedule and the mapping matrices are used in *CalcReliability* routine to compute the different reliability values. Specifically, the schedule is used to compute the lifetime reliability, R_P^{temp}, considering the thermal impact as detailed in Chapter 3; the mapping matrices are used to compute the reliability considering checkpointing for transient fault-tolerance, R_T^{temp}, as explained in Sections 4.2.2 and 4.2.3. Finally, the wear-out minimum initial mapping is the optimization technique proposed in Chapter 3.

4.3 Reliability-Aware Co-Design

Figure 4.6 shows the reliability-aware hardware-software co-design approach proposed in this work. The approach consists of two components – reliability-aware design space exploration ($RDSE$) with individual applications and reliability-aware Pareto merging for use-cases. The $RDSE$ generates a set of Pareto-points (actor distribution and checkpoint assignment), which are optimal in terms of reliability and resource usage (i.e. the number of processing cores and size of the reconfigurable area). Next, analysis is performed with the given set of use-cases. Specifically, an ILP-based technique is proposed to merge the Pareto-points (obtained in the previous step) of the applications constituting an use-case to determine the minimum resource usage while satisfying the given energy and performance budget.

4.3.1 Design Metrics

Performance: The minimum throughput required for an SDFG is denoted by $\mathbb{T}_c$. This is the performance parameter used in the approach and is considered a design constraint.

Energy: The energy consumption of the multiprocessor system is modeled in Chapter 3. The energy metric is used as a design constraint. Specifically, $E^{tot} \leq E^{max}$, where E^{max} is the given energy budget.

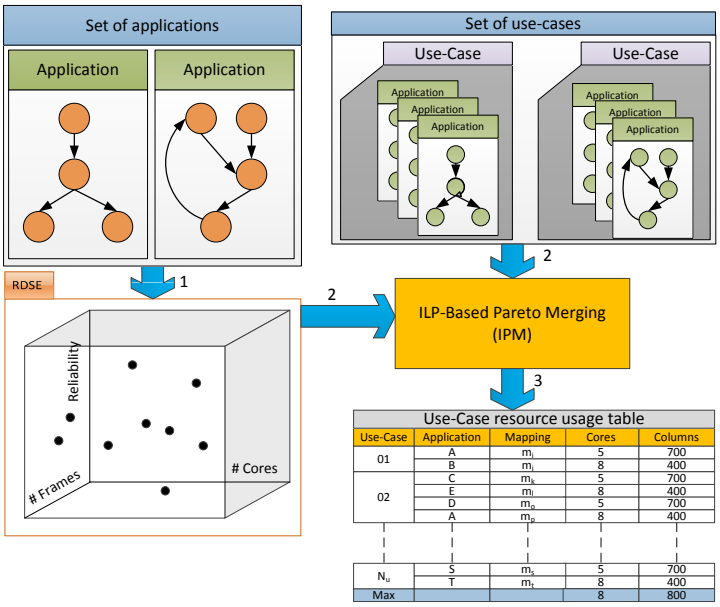


Figure 4.6: Proposed hardware-software co-design flow.

Algorithm 8 $RDSE()$: Design space exploration for reliability-resource trade-off

Input: $\mathcal{G}_{app}$, $\mathcal{G}_{arc}$, $\mathbb{T}_c$, E^{max}, N_c^{max}, RA^{max}
Output: Mapping Pareto database $ParetoDB$

```
for N_c ∈ [1..N_c^max] do
    for {S_RA = e; S_RA ≤ RA^max; S_RA = S_RA + e} do
        (M_d  M_c) = HSAP(G_app, G_arc^{N_c+1}, T_c, S_RA) //Hardware-software actor partitioning.
        E^tot = CalculateTotalEnergy(M_d) //Calculate the total energy of an application.
        [S  T] = MSDF^3(M_d, G_app, G_arc^{N_c+1}) //Calculate the schedule and throughput.
        M = CalculateMTTF(S, M_d, M_c) //Calculate the MTTF using Equation 4.13.
        if T ≥ T_c && E^tot ≤ E^max then
            ParetoDB(N_c, S_RA).push(M_d, M_c, M_J, E^tot) //Store in a database.
        end if
    end for
end for
Process ParetoDB to retain only Pareto-optimal points.
```

Cost: The design cost is specified in terms of the number of cores and the size of the reconfigurable area. The maximum number of cores allowed is denoted by N_c^{max} and maximum size of the reconfigurable area is denoted by RA^{max}. This is used as a design constraint.

Reliability: The reliability considering wear-out and transient faults are modeled in Equations 2.12 and 4.14, respectively. The co-design problem is demonstrated here to maximize the lifetime reliability (measured as mean time to failure) under a given constraint of transient fault-tolerance, specified as mean time between failure. However, the technique can be easily adopted to optimize for these parameters (MTTF or MTBF) individually, or combined together in the form of a joint metric.

4.3.2 Reliability-Resource Usage Trade-off

Algorithm 8 provides the pseudo-code of the reliability-aware design space exploration to determine the reliability-resource trade-off. As shown in Section 4.1 (refer to Figure 4.1), the given problem can be solved in a hierarchical manner with the reliability-aware hardware-software actor partitioning on a given reconfigurable multiprocessor system (i.e with N_c cores where $N_c \leq N_c^{max}$ and RA frames where $RA \leq RA^{max}$) at the lower hierarchy, followed by its integration in the higher level problem of determining the reliability-resource trade-off. An epoch ($e = 100$) is defined and the frames allocated to the multiprocessor system is incremented by e at each iteration. The algorithm inputs the application graph and the design constraints – the throughput constraint $\mathbb{T}_c$, the energy budget E^{max}, and the design cost N_c^{max} and RA^{max}. The algorithm then determines the actor distribution and the MTTF value for every combination of core count and reconfigurable area frames by invoking the $HSAP$ subroutine (Algorithm 6). The energy and the schedule are computed (lines 4 - 5). The schedule and the actor distribution are used to compute the MTTF (Equation 4.13). If the energy is lower than the energy budget and the throughput constraint is satisfied, the actor distribution, the MTTF, and the energy value are stored in the Pareto database $ParetoDB$ corresponding to the number of cores and size of the reconfigurable area (lines 7 - 9). Finally, the $ParetoDB$ is processed to retain only the Pareto-optimal points.

4.3.3 ILP-Based Pareto Merging

The objective in the next stage of the co-design is to determine the reliability (MTTF) maximum mappings for applications enabled simultaneously. These mappings determine the size of the platform. The problem is formulated as a binary integer linear programming (ILP) problem and is solved using Matlab. The inputs to this step are the set of mappings (from $ParetoDB$) for every application of a given use-case. To limit the scalability of this approach (the ILP), only the Pareto-optimal points (i.e. the actor distribution and checkpoint assignments that are optimum with respect to reliability, energy, number of cores, and size of reconfigurable area) are used. Every Pareto-point of an application is associated with four parameters – number of cores, reconfigurable area size, reliability and energy. This step selects one Pareto-point for every application of a use-case such that the reliability is maximized while satisfying the design cost

and energy budgets. To aid the understanding of this approach, the ILP is formulated for a two application use-case. The technique can be trivially extended to include any multi-application use-cases.

Let *App_A* and *App_B* be the two applications constituting a use-case. The following notations are defined for the ease of problem formulation.

$$
\begin{aligned}
N_X &= \text{number of Pareto-points of } App\_X \text{ where X = A or B} \\
m_1^X, \cdots, m_{N_X}^X &= N_X \text{ Pareto-points of } App\_X \\
n_i^X &= \text{number of cores of Pareto-point } m_i^X \text{ of application } App\_X \\
c_i^X &= \text{reconfigurable area usage of Pareto-point } m_i^X \\
r_i^X &= \text{reliability of Pareto-point } m_i^X \\
e_i^X &= \text{energy consumption of Pareto-point } m_i^X
\end{aligned}
$$

Let Y_{ij} is defined as follows.

$$
Y_{ij} = \begin{cases} 1 & \text{if } m_i^A \text{ and } m_j^B \text{ are selected} \\ 0 & \text{otherwise} \end{cases} \tag{4.16}
$$

Constraints of the ILP

- One Pareto-point for each application is to be selected i.e., $\sum_{i=1}^{N_A} \sum_{j=1}^{N_B} Y_{ij} = 1$
- Design cost is to be satisfied i.e.,

$$
\sum_{i=1}^{N_A} \sum_{j=1}^{N_B} Y_{ij} \cdot (n_i^A + n_i^B) \leq N_c^{max} \text{ and } \sum_{i=1}^{N_A} \sum_{j=1}^{N_B} Y_{ij} \cdot (c_i^A + c_i^B) \leq RA^{max}
$$

- Energy budget is to be satisfied i.e., $\sum_{i=1}^{N_A} \sum_{j=1}^{N_B} Y_{ij} \cdot (e_i^A + e_i^B) \leq E^{max}$

Objective of the ILP

The reliability of the platform with applications A and B executing simultaneously is

$$
\begin{aligned}
R_A &= \text{ Reliability due to A } = \sum_{i=1}^{N_A} \sum_{j=1}^{N_B} Y_{ij} \cdot r_i^A \\
R_B &= \text{ Reliability due to B } = \sum_{i=1}^{N_A} \sum_{j=1}^{N_B} Y_{ij} \cdot r_i^B
\end{aligned} \tag{4.17}
$$

The overall reliability optimization objective is to maximize $\min\{R_A, R_B\}$.

ILP Solution

Let the solution of the ILP be represented as $Y_{ij} = 1$ for $i = i_b$ and $j = j_b$, and $Y_{ij} = 0$ otherwise. The resources used for this use-case (with applications A and B) are as follows: number of cores = $(n^A_{i_b} + n^B_{j_b})$ and reconfigurable area size = $(c^A_{i_b} + c^B_{j_b})$. Thus, the ILP is solved for all use-cases to fill the use-case resource table of Figure 4.6. A point to be noted is that, the execution time to fill the use-case resource table is strongly dependent on the number of use-cases. As established in [28], the number of use-cases grows exponentially with the number of applications. Use-case pruning techniques proposed in [28] are adopted here. Once the use-case resource table is filled, the final platform is determined as the highest number of cores and reconfigurable area size.

4.4 Results

All proposed algorithms are coded in C++. Fifty synthetic SDFGs are generated from [138] with number of actors between 8 and 32. Additionally, a set of real-life applications (streaming and non-streaming) are considered from [138, 142]. These applications are *H.263 Encoder/Decoder*, *H.264 Encoder*, *MP3 Decoder*, *MPEG4 Decoder*, *JPEG Decoder*, *Sample Rate Converter*, *FFT*, *iFFT* and *Romberg Integration*. The hardware-software task partitioning experiments (Sections 4.4.2 - 4.4.5) are conducted on a multiprocessor system with 9 homogeneous cores and 500 frames of reconfigurable area.

4.4.1 Algorithm Complexity

The complexity of Algorithm 6 is computed as follows. Let η be the average number of actors that can be accommodated in the given reconfigurable area constraint. There are therefore η iterations of the outer while loop (lines 2 - 10). At each iteration, lines 4 - 6 are executed for all actors in $\mathbb{A} \setminus HList$. This can be upper bounded by N_a, the total number of actors in the application graph. Line 8 finds the maximum from a list of N_a elements. The complexity of Algorithm 6 is given by

$$C_6 = O\left(\eta \cdot (N_a \cdot O(FindMinRG) + N_a)\right) = O\left(N_a^2 \cdot C_7\right) \tag{4.18}$$

where $O(FingMinRG) = C_7$ is the complexity of the $FindMinRG$ routine and $\eta \leq N_a$. The complexity of the $FindMinRG$ routine is computed as follows. The $MSDF^3$ engine computes the schedule starting from a given actor distribution. This can be performed

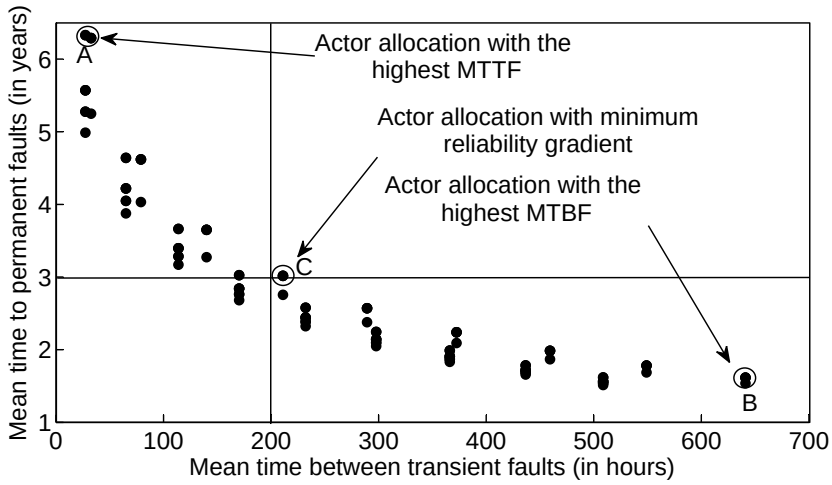


Figure 4.7: Mean time to failure for transient and permanent faults for h.263 decoder.

in $O\left(N_a \log N_a + N_a \cdot \pounds\right)$ (ref. [77]) where $\pounds$ is the average number of successors of an actor. The reliability gradient can be computed in $O\left(N_a + N_c\right)$. Assuming the outer while loop executes for χ times on average, the complexity of Algorithm 7 is

$$C_7 = O\left(\chi \cdot N_a \cdot N_c \cdot N_{cp} \cdot \left(N_a \log N_a + N_a \cdot \pounds + N_a + N_c\right)\right) = O\left(N_a^4 \cdot N_{cp}\right) \tag{4.19}$$

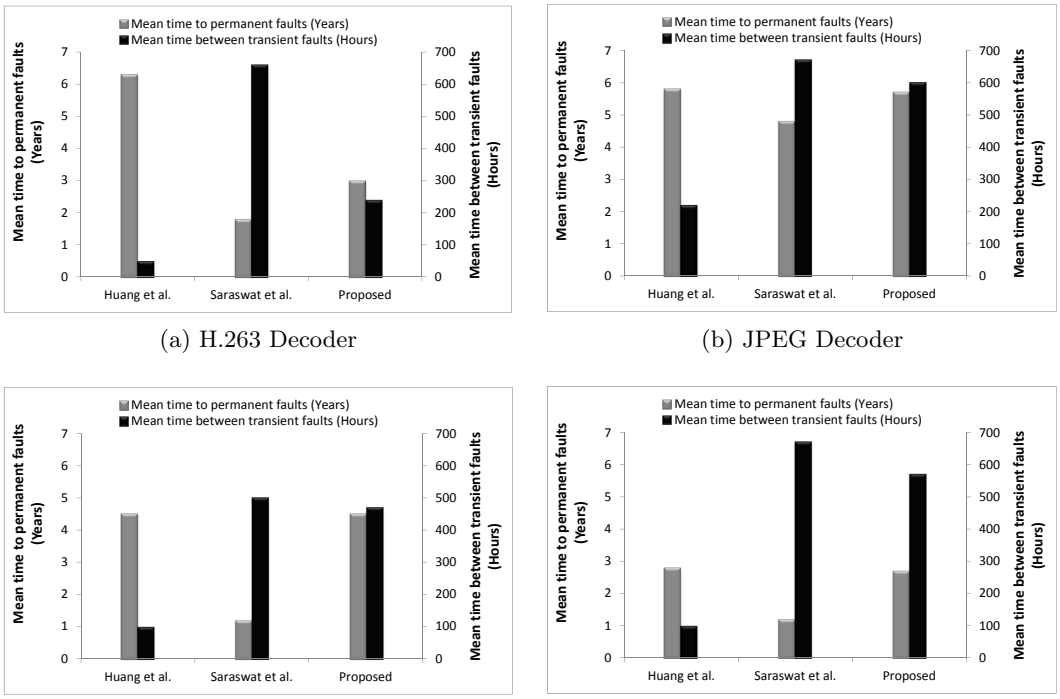
using $\pounds \leq N_a$ and $N_c \leq N_a$ and N_{cp} is the maximum number of checkpoints per actor. Combining Equations 4.18 and 4.19, the complexity of Algorithm 6 is given by

$$C_6 = O\left(N_a^6 \cdot N_{cp}\right) \tag{4.20}$$

4.4.2 Reliability Trade-off Results

Figure 4.7 plots the mean time to failures considering wear-out (MTTF) and mean time between transient faults (MTBF) for different design solutions (actor distribution and checkpoint assignment) obtained from the proposed $HSAP$ algorithm for H.263 Decoder application. The figure also plots the solution obtained using the wear-out-aware task mapping technique of [95] (marked in the figure by the alphabet A) and the checkpointing based transient fault-tolerant technique of [44] (marked in the figure by the alphabet B). For demonstration purpose, the transient fault arrival rate of 1 fault every 100 hours of operation is considered. The reliability requirements are set as follows: MTTF = 3 years and MTBF = 200 hours, and are shown in the figure by the solid lines.

As established previously, the wear-out aware task mapping technique of [95] does not consider transient fault-tolerance, and therefore the MTBF reported in the figure is obtained by considering zero checkpoints in the actor execution. On the other hand,



(a) H.263 Decoder

(b) JPEG Decoder

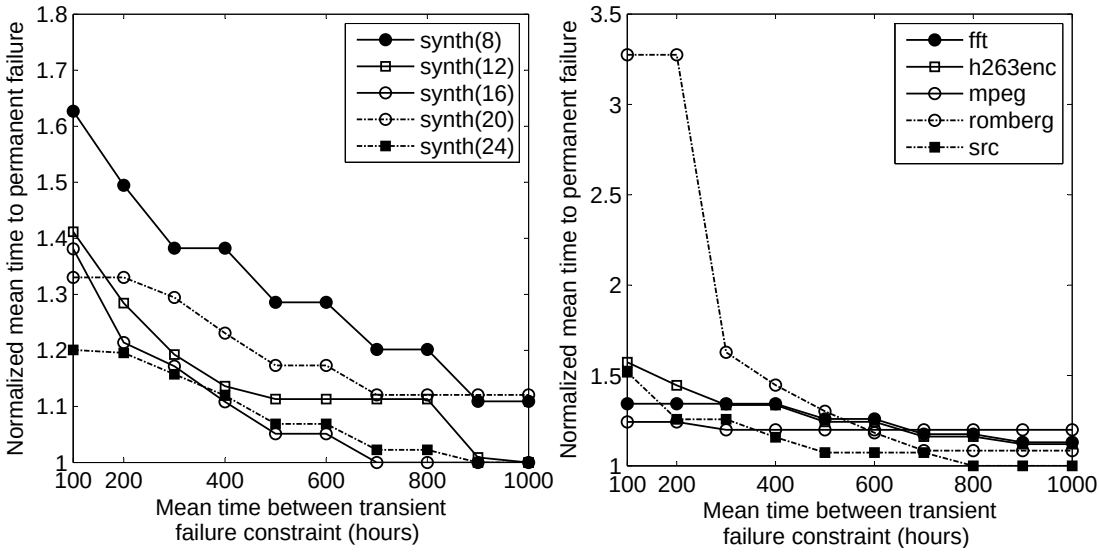
(c) Inverse FFT

(d) MP3 Decoder

Figure 4.8: Reliability trade-off results for four different applications.

the checkpoint-based transient fault-tolerance technique of [44] does not consider lifetime reliability. The number of checkpoints for the actors are selected such that the reliability is maximized (refer to Figure 4.5). The proposed $HSAP$ algorithm selects the point marked C in the figure. This point satisfies the reliability requirements for both fault types and results in minimum reliability gradient i.e. the minimum degradation of lifetime reliability considering wear-out with a maximum increase in the reliability considering transient faults. The actor distribution corresponding to this point improves the MTTF by 100% (2x improvement) as compared to that obtained using [44].

To give more insight into the different trade-off trends obtained for different applications, Figure 4.8 plots the trade-off between the two reliabilities for four applications, including the H.263 decoder application result shown in Figure 4.7. The result obtained using the proposed technique is compared with [95] and [44]. The MTTF and the MTBF constraint for these applications are set to 3 years and 20 hours, respectively. The plot for the JPEG decoder (Figure 4.8b) shows that the MTTF and the MTBF requirements are satisfied by all the three techniques. However, the proposed technique and the technique of [44] offer better reliability trade-offs than [95]. Results for this application show that the improvement in the proposed approach is insignificant – 18% higher MTTF and



(a) Synthetic applications (b) Real applications

Figure 4.9: Normalized MTTF with varying transient fault-tolerance constraint.

10% lower MTBF as compared to [44]. Therefore, both the proposed approach and that of [44] are good solutions, and either of them can be selected as the final solution.

The application inverse FFT (Figure 4.8c) demonstrate similar trend as H.263 decoder. Out of the three techniques, only the proposed technique satisfies both the MTTF and MTBF requirements. Therefore as discussed previously, only the proposed solution is the valid one. Finally, for the application MP3 decoder (Figure 4.8d), the MTTF requirement is violated by all the techniques. This is due to the high stress introduced in the system due to the large number of actors. The MTBF requirement is violated by [95], but both the proposed and the [44] satisfy the MTBF requirement. For this application, the proposed technique results in 125% higher lifetime with less than 15% lower MTBF as compared to [44], clearly demonstrating its superiority. Out of all the 60 applications (real-life and synthetic) considered, 6 applications (2 real-life and 4 synthetic) show trends similar to the JPEG decoder and other 3 applications (1 real-life and 2 synthetic) that of the MP3 decoder. The remaining 51 applications (7 real-life and 44 synthetic) show trends similar to H.263 decoder.

To summarize the results for all these sixty applications, the proposed technique improves the platform lifetime by 18% to 225% (with an average of 60%) as compared to the highest MTBF technique of [44]. Finally, the MTTF using the proposed technique is within an average 15% of the highest MTTF of [95].

4.4.3 MTTF Results with Varying MTBF Constraint

Figure 4.9 plots the normalized lifetime (i.e., MTTF) obtained using the proposed technique with varying transient fault-tolerance constraint for five synthetic and five real-life applications. The synthetic applications are labeled as *synth(n)* where n denotes the number of tasks of the application; the five real-life applications are *FFT, H.263 Encoder, MPEG4 Decoder, Romberg Integration*, and *Sample Rate Converter*. The MTTF obtained using the proposed technique is normalized with respect to that obtained using the transient fault-tolerant technique of [44]. The transient fault-tolerance constraint is specified as MTBF, and is varied from 100 hours to 1000 hours. The constraint is interpreted as follows: MTBF requirement of say, 100 hours implies that with the given fault arrival rate, the system should be capable of correcting these faults using checkpointing mechanism, with the time allowed between two non-correctable faults to be 100 hours. Clearly, higher the MTBF requirement, the more stringent is the transient fault-tolerance constraint. The range for the MTBF constraint (100 to 1000 hours) represents the varying reliability requirement of both safety and non-safety critical applications.

An interesting trend to observe from these figures is that, as the transient fault-tolerance constraint becomes more and more stringent (higher MTBF requirement), the normalized MTTF drops. This is expected due to conflicting nature of the two fault types as established in Section 4.2. For critical applications, where high reliability is desired (e.g. non-correctable faults every 1000 hours of operation), the MTTF obtained using the proposed technique for applications, such as *synth(12), synth(16), synth(24)*, and *Sample Rate Converter*, is similar to the MTTF values obtained using [44] (normalized MTTF close to one). For the remaining six applications, the proposed technique performs better at this MTBF requirement. On average for all the sixty applications considered, the proposed technique outperforms the existing technique by achieving 10% higher MTTF even at a high transient fault-tolerance requirement of 1000 hours. These results suggest that the existing transient fault-tolerant techniques leave a significant scope of lifetime improvement, which is addressed by the proposed technique. On the other hand, for less stringent reliability requirement of 100 hours, the proposed technique provides an average 60% lifetime improvement. These results show the need for considering processor wear-out in the checkpoint selection for transient fault-tolerance.

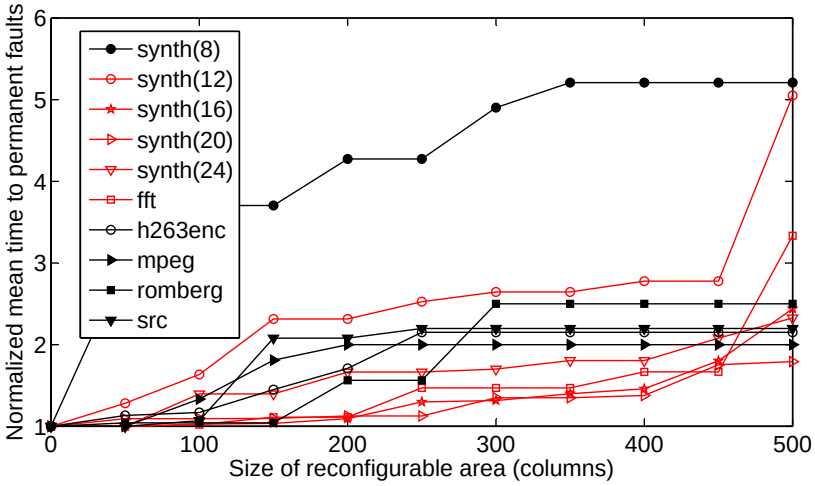


Figure 4.10: MTTF and size of reconfigurable area trade-off.

4.4.4 Reliability and Reconfigurable Area Trade-off

An important trade-off analysis for reconfigurable multiprocessor systems is to determine the gain in reliability with increase in reconfigurable area i.e., reliability and reconfigurable area trade-off results. To demonstrate this, an experiment is conducted with the same set of ten applications (real-life and synthetic) using the proposed technique, and the reliability results are compared with the reliability obtained with no reconfigurable area i.e., on a static multiprocessor system with the same number of cores as the reconfigurable one. Figure 4.10 plots the normalized MTTF as the size of the reconfigurable area is increased. The MTTF obtained using the proposed $HSAP$ algorithm for an application is normalized with respect to the MTTF obtained without reconfigurable area. Few trends can be observed from the figure. First, the MTTF (i.e. reliability) improves with an increase in the size of the reconfigurable area. This is because, with increase in the size of the reconfigurable area, more actors can be implemented as hardware. This reduces the stress on the processing cores leading to an increase in lifetime reliability.

Second, the improvement in MTTF saturates beyond a certain reconfigurable area size. This is due to the limited lifetime improvement possible after remapping most of the actors of an application. For some applications, such as *Sample Rate Converter* and *H263 Encoder*, the saturation point is at lower size of the reconfigurable area. These applications are marked in the figure by black solid lines. For other applications, the saturation point is beyond 500 columns. Third, for some applications, such as *synth(16), synth(20)* and *FFT*, the improvement of lifetime is slower up to 300 columns; significant improvement is observed as the reconfigurable area size is increased beyond 300 columns.

Table 4.1: Execution time (in sec) of the $HSAP$ algorithm.

Actors	RA size = 100				RA size = 300			
	cores = 2	cores = 4	cores = 6	cores = 8	cores = 2	cores = 4	cores = 6	cores = 8
8	50	75	100	125	90	145	150	160
16	150	670	720	775	500	1,140	2,140	3,000
24	785	1,860	3,300	5,575	2,580	7,100	12,150	13,560
32	1,140	3,020	5,130	6,790	2,850	7,500	12,700	14,180

For other applications, such as *synth(8), synth(12)* and *Sample Rate Converter*, lifetime improvement is possible even with small reconfigurable area.

The conclusion to derive from these results is that, different applications exhibit different trade-offs with respect to reconfigurable area and lifetime performance. It is essential to characterize each application during the design phase to explore such trade-off. This knowledge can be applied at run-time during application mapping and reconfigurable area distribution among multiple simultaneous applications. As an example, if the reconfigurable area available at a given time during operation is 100 frames and application *FFT* and *Sample Rate Converter* needs to be mapped, it is beneficial to reserve the reconfigurable area for *Sample Rate Converter* that provides significant improvement in lifetime than *FFT*.

4.4.5 Execution Time of the HSAP Algorithm

Table 4.1 reports the execution time of the proposed $HSAP$ algorithm as the number of actors and cores are scaled for reconfigurable area of 100 and 300 frames. As seen from this table, the execution time increases with increase in the number of actors and cores. However, the time growth can be accommodated as the analysis are performed at design-time. Moreover, with an increase in the size of the reconfigurable area, the execution time also increases. Two factors contribute to this: first, with increase in the reconfigurable area size, the number of iterations of the outer while loop (lines 2 - 10) of Algorithm 6 increases; second, at each iteration, the number of actors to be analyzed i.e. the iterations of Algorithm 7 reduces as more actors are marked as hardware actors.

A point to note is that, the optimization problem formulated in Sections 4.2 can be solved directly using standard solver e.g., CPLEX. However, the execution time grows exponentially with the number of actors and cores. The solver fails to provide results beyond 8 actors mapped on 6 cores even after running for more than 12 hours. For applications for which the optimization terminates within the given time frame of 12

Table 4.2: Platform determination with area, energy and reliability results.

single applications						use-cases					
Applications	Technique	Cores	RA Size	Energy	MTTF	Applications	Technique	Cores	RA Size	Energy	MTTF
MPEG4 Decoder	*ECosynth*	9	200	0.75	0.56	*usecase_1*	*ECosynth*	16	700	0.81	0.51
	RPGen	16	0	1.00	0.73		*RPGen*	16	0	1.00	0.58
	Proposed	8	300	0.90	0.94		Proposed	12	600	1.00	0.90
JPEG Decoder	*ECosynth*	2	100	0.79	0.68	*usecase_2*	*ECosynth*	10	1000	0.85	0.60
	RPGen	4	0	0.96	0.81		*RPGen*	16	0	1.00	0.68
	Proposed	4	200	0.91	1.00		Proposed	10	800	0.90	1.00
final platform	*ECosynth*	9	600	–	–	final platform	*ECosynth*	16	1000	–	–
	RPGen	16	0	–	–		*RPGen*	16	0	–	–
	Proposed	8	500	–	–		Proposed	12	800	–	–

hours, the proposed $HSAP$ algorithm provides up to 500× reduction in execution time.

4.4.6 Hardware-Software Co-Design Results

To establish the area and energy overhead introduced for reliability aware co-design, the proposed technique is compared with energy-aware co-synthesis technique of [38] (referred to as *ECosynth*) and the reliability-aware platform generation technique of [116] (referred to as *RPGen*). It is to be noted that the size of platform for *ECosynth* is determined considering energy only, however the platform reliability value is determined using the $HSAP$ algorithm proposed in Section 4.2. Table 4.2 reports the resource usage (in terms of the number of cores and reconfigurable area), normalized energy and reliability of the proposed technique in comparison with *ECosynth* and *RPGen* techniques for single applications as well as for use-cases. The design cost for the reconfigurable multiprocessor system is set to a maximum of 16 cores and 1000 reconfigurable area columns. The reference for reliability is obtained by mapping an application on this highest architecture (with 16 cores and 1000 columns) using the proposed $HSAP$ algorithm. The energy reference is the energy budget E^{max}.

Single Application Results:

For all single applications considered, the *ECosynth* results in the least energy consumption (for energy consumption, lower is better). This is expected as the primary objective of this technique is to minimize energy. However, the lifetime (measured as MTTF) obtained using this technique is the least (for MTTF, higher is better). For the *MPEG4 decoder* application, the *ECosynth* results in 44% lower MTTF (row 3, column 6) and 25% lower energy (row 3, column 5) as compared to the reference approach. The *RPGen*

technique determines the number of cores based on reliability with energy as a constraint. This technique does not consider reconfigurable area, and therefore the entries reporting the reconfigurable area usage (row 4, columns 4) is 0. The lifetime reliability using this technique is on average better than *ECosynth* by 30% as this is explicitly maximized. The energy consumption of this technique is, however, higher by 33%. This high energy overhead of *RPGen* is attributed to the fact that the underlying architecture for this technique is the static multiprocessor system with no reconfigurable area. Therefore, the technique does not benefit from the lower energy consumption of the hardware implementation of actors. In comparison to both these techniques, the proposed technique results in the highest MTTF with an improvement of 28% with respect to *RPGen* and 67% with respect to *ECosynth*. In terms of energy consumption, the proposed technique consumes 10% lower energy than *RPGen*,. Finally, the resource utilization of the proposed technique is also better than the existing techniques. A similar trend is observed for all the single applications considered (including those not shown in this table). On average for all these applications, the proposed technique improves lifetime by an average 30% with respect to *RPGen* and 65% with respect to *ECosynth*.

The minimum resources required for all the single applications are reported in the table at rows 9-11, columns 3-4. As seen from these entries, the proposed approach satisfies the resource constraint and also leads to the minimum resource usage.

Use-case Results:

To demonstrate the performance of the ILP-based Pareto merging technique for use-cases, an experiment is conducted with ten synthetic use-cases. Two of these are reported in the table. The compositions of these use-cases are as follows: $usecase\_1 = \langle JPEGDec, MP3Dec \rangle$ and $usecase\_2 = \langle MPEG4Dec, SRC, H263Dec \rangle$. Since none of the two existing works consider use-cases for platform determination, the resource usage for these techniques is performed by optimizing the concurrent applications individually, with the resource constraint modified according to their throughput requirements. An example is illustrated below.

Let $usecase\_i = \langle App\_A, App\_B \rangle$ with throughput constraint of App_A is 1.5 times the throughput constraint of App_B. The overall resource constraint of 16 cores with 1000 reconfigurable columns is distributed to these applications such that the constraint

on *App_A* is 9 cores with 600 columns, and that of application *App_B* is 7 cores with 400 columns. The proposed approach uses ILP based Pareto merging to determine the resource requirement for every use-case. As seen from the table, for *usecase_1*, the *ECosynth* still achieves the minimum energy consumption. The MTTF is still the least. The *RPGen* although maximizes MTTF, the improvement over that of *ECosynth* is only 13%. This is due to the non-availability of the reconfigurable area leading to the missing reliability and reconfigurable area trade-off. The proposed approach improves lifetime by 55% as compared to *RPGen*. A similar trend is observed for the second use-case. On average for all the ten use-cases considered, the proposed approach improves lifetime by 50% with respect to *RPGen* and 70% with respect to *ECosynth*, while satisfying the given area, energy and performance budget.

The maximum resource used for all the ten use-cases (including the two shown in the table) is reported in rows 9-11, columns 9-10. As seen from these results, the proposed approach minimizes the number of processing cores by 25% and reconfigurable area usage by 20% with respect *ECosynth*, while maximizing the lifetime by an average 65% for single application and an average 70% for use-cases and satisfying the area, energy, and performance constraint. With respect to the lifetime maximum technique (*RPGen*), the proposed co-design approach improves lifetime by an average 30% for single applications and an average 50% for use-cases.

4.5 Remarks

This work presented a fast heuristic for hardware-software task partitioning for reconfigurable multiprocessor systems. The objective is to improve the transient fault-tolerance of the system together with the lifetime reliability of the cores. Based on this heuristic, a hardware-software co-design technique is proposed that determines the minimum resources needed to maximize the reliability while satisfying the given energy, cost, and performance constraint. The co-design methodology incorporates integer linear programming to merge the Pareto-points of the individual applications to determine the resource usage for concurrent applications (use-cases). Experiments conducted with synthetic and real-life application SDFGs demonstrate that the proposed hardware-software partitioning technique maximizes lifetime reliability by 10% for a stringent transient fault-

tolerance requirement. With relaxed requirements, the proposed approach is able to improve lifetime by an average 60%. Moreover, the proposed hardware-software co-design approach improves the lifetime reliability by an average 70%, while consuming 25% fewer cores and 20% lower reconfigurable area size as compared to the existing technique.

CHAPTER 5

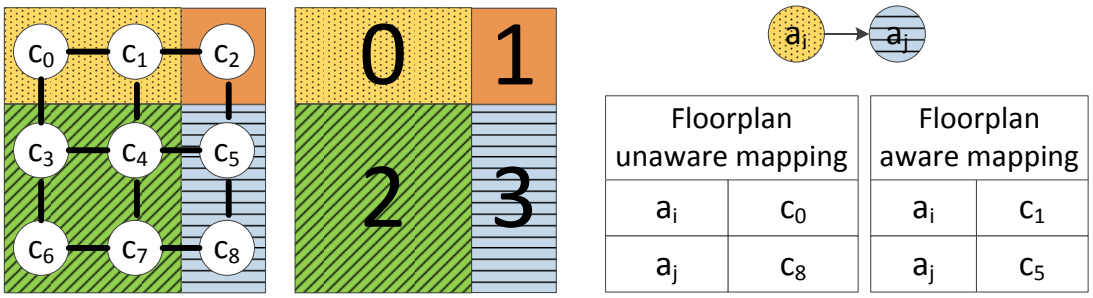
Design-time Analysis for Fault-Tolerance

5.1 Introduction

One of the highly desired features of modern multiprocessor systems is fault-tolerance i.e., the ability of these systems to continue operation in the presence of faults, albeit an acceptable performance degradation. This work attempts to solve the following problem. Given a heterogeneous multiprocessor architecture and a set of multimedia and other high-performance embedded applications, how to assign and order the tasks of every application on the component cores such that the total energy consumption (computation and communication) is minimized while guaranteeing to satisfy the performance requirement (e.g., throughput) of these application under all possible core fault-scenarios. The scope of this work is limited to permanent faults of cores. It assumes a given multiprocessor architecture (floorplan), and therefore the selection of cores (number and/or types) for the architecture and their placement (coordinates) are not addressed.

Following are the key contributions of this work:

- a fault-aware task mapping technique to minimize the computation and communication energy while satisfying the application throughput requirement;
- a scheduling technique to minimize the run-time schedule construction and schedule



(a) Floorplan of the multiprocessor system.

(b) Impact of floorplan unaware mapping.

Figure 5.1: Architecture model.

storage overhead; and

- a heuristic to minimize the design space exploration time.

The remainder of this chapter is organized as follows. The problem formulation is discussed in Section 5.2, followed by the proposed design methodology in Section 5.3. The energy minimum task mapping technique for different fault-scenarios is discussed next in Section 5.4 and the proposed self-timed execution-based scheduling technique in Section 5.5. Experimental setup and results are discussed in Section 5.6. Lastly, conclusions are presented in Section 5.7.

5.2 Problem Formulation

5.2.1 Application and Architecture Model

An application is represented as synchronous data flow graphs (SDFGs) $\mathcal{G}_{app} = (\mathbb{A}, \mathcal{C})$ consisting of a finite set $\mathbb{A}$ of actors and a finite set $\mathcal{C}$ of channels. Every actor $\boldsymbol{a}_i \in \mathbb{A}$ is a tuple (t_i, μ_i), where t_i is the execution time of $\boldsymbol{a}_i$ and μ_i is its state space (program and data memory). The number of actors in an SDFG is denoted by N_a where $N_a = |\mathbb{A}|$. The performance of an SDFG is specified in terms of throughput constraint $\mathbb{T}_c$.

The architecture consists of processing cores interconnected in a mesh-based topology as shown in Figure 5.1a. The different zones in the figure represent heterogeneity with the cores within each zone being homogeneous. In all existing studies on reactive fault-tolerance, floorplan of the underlying multiprocessor platform is ignored, that is, heterogeneous cores are considered without their actual coordinates. This can impact communication energy, as shown in Figure 5.1b. Here, actors $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$ require core types 0 and 1, respectively. Floorplan-unaware and floorplan-aware mapping examples

are provided in the two tables. Clearly, floorplan-unaware mapping can lead to higher data communication energy (i.e., data communicated over four hops between $\boldsymbol{c}_0$ and $\boldsymbol{c}_8$ as compared to 2 hops between $\boldsymbol{c}_1$ and $\boldsymbol{c}_5$ in floorplan-aware mapping).

The architecture is represented as a graph $\mathcal{G}_{arc} = (\mathbb{C}, \mathbb{E})$, where $\mathbb{C}$ is the set of nodes representing cores of the architecture and $\mathbb{E}$ is the set of edges representing communication channels among the cores. The total number of cores is denoted by N_c i.e., $N_c = |\mathbb{C}|$. Each core $\boldsymbol{c}_j \in \mathbb{C}$ is a tuple $\langle h_j, O_j \rangle$, where h_j represents the heterogeneity type of $\boldsymbol{c}_j$, and O_j is the set of operating points (voltages and frequencies) supported on $\boldsymbol{c}_j$.

5.2.2 Mapping Representation

The following notations are defined.

N_o	maximum number of operating points of a core
M_n	mapping of $\mathcal{G}_{app}$ on $\mathcal{G}_{arc}$ with n cores where $n \leq N_c$
ϕ_i	core on which actor $\boldsymbol{a}_i$ is mapped in mapping M_n
θ_i	frequency assigned to actor $\boldsymbol{a}_i$
Ψ_j	set of actors mapped to core $\boldsymbol{c}_j$
s_f	fault-scenario with f faulty cores $= \langle \boldsymbol{c}_{i_1}, \boldsymbol{c}_{i_2}, \cdots, \boldsymbol{c}_{i_f} \rangle$

The objective of the optimization problem is to minimize energy consumption for all fault-scenarios by solving the following:

- *actor distribution:* i.e., to determine the assignment of the actors of the SDFG on the cores of the multiprocessor system;
- *operating point:* i.e., to determine the voltage and frequency of the cores for executing the actors of the SDFG.

The mapping is represented as $M_n = \begin{pmatrix} \mathcal{M}_d^n & \mathcal{M}_o \end{pmatrix}$. Two variables $x_{i,j}$ (representing the *actor distribution*) and $y_{i,k}$ (representing the *operating point*) are defined as follows.

$$x_{i,j} = \begin{cases} 1 & \text{if actor } \boldsymbol{a}_i \text{ is executed on core } \boldsymbol{c}_j \\ 0 & \text{otherwise} \end{cases} \qquad y_{i,k} = \begin{cases} 1 & \text{if actor } \boldsymbol{a}_i \text{ is executed at operating point } o_k \\ 0 & \text{otherwise} \end{cases}$$

Constraints on these variables are set such that an actor is mapped to only one core at a single operating point. Thus,

$$\sum_{j=0}^{N_c-1} x_{i,j} = 1 \text{ and } \sum_{k=0}^{N_o-1} y_{i,k} = 1 \quad \forall \boldsymbol{a}_i \in \mathbb{A} \tag{5.1}$$

The *actor distribution* and *operating point* of an SDFG are represented as two matrices:

$$\mathcal{M}_d^n = \begin{pmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,n-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N_a-1,0} & x_{N_a-1,1} & \cdots & x_{N_a-1,n-1} \end{pmatrix} \text{ and } \mathcal{M}_o = \begin{pmatrix} y_{0,0} & y_{0,1} & \cdots & y_{0,N_o-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,N_o-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N_a-1,0} & y_{N_a-1,1} & \cdots & y_{N_a-1,N_o-1} \end{pmatrix} \tag{5.2}$$

The core assignment and operating point of actor $\boldsymbol{a}_i$ are given by

$$\phi_i = \mathbf{X}_i \times \mathbb{N}_{N_c} \text{ where } \mathbf{X}_i = \begin{pmatrix} x_{i,0} & x_{i,1} & \cdots & x_{i,N_c-1} \end{pmatrix} \text{ and } \mathbb{N}_{N_c} = \begin{pmatrix} 0 & 1 & \cdots & N_c-1 \end{pmatrix}^T$$
$$\theta_i = \mathbf{Y}_i \times \mathbb{N}_{N_o} \text{ where } \mathbf{Y}_i = \begin{pmatrix} y_{i,0} & y_{i,1} & \cdots & y_{i,N_o-1} \end{pmatrix} \text{ and } \mathbb{N}_{N_o} = \begin{pmatrix} 0 & 1 & \cdots & N_o-1 \end{pmatrix}^T$$

5.2.3 Mapping Encoding

An ID is assigned to each mapping M_n as calculated in Equation 5.3.

$$mID(M_n) = \sum_{i=1}^{N_a} \phi_i \cdot (N_c)^i \tag{5.3}$$

5.2.4 Energy Modeling

The computation and the communication energy are modeled in Chapter 3; this section provides the modeling of the energy associated with task-migration. Migration overhead associated with moving from one mapping to another is governed by two quantities – the state space of the actors(s) participating in the migration process and the distance (hops) through which the state space is migrated[1]. It is assumed that a given multiprocessor system consists of one or more **task migration modules** (TMMs), which can access the memory of a core without interfering with its operation. For these systems, the state space of an actor (on a faulty core) can be recovered and hence migrated to some other core. For multiprocessor systems without TMMs, task migration involves migrating the state space of an actor from the main memory to the new core where it is to be mapped. To better couple with the computation and the communication energy, the migration overhead is represented as energy and is termed as **migration energy**.

$$MigrationEnergy = \sum_{\forall \boldsymbol{a}_i \in \Psi_j} S_i \cdot E_{bit}(\phi_i^{init}, \phi_i^{final}) \tag{5.4}$$

[1]The state space of an actor consists of the the data memory and the pre-compiled object code for the h different core types.

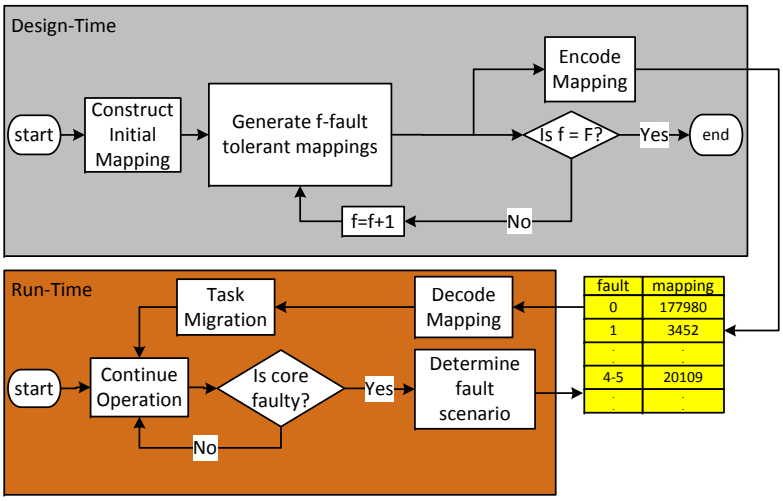


Figure 5.2: Proposed Design Methodology.

where S_i is the state space of actor $\boldsymbol{a}_i$, ϕ_i^{init} and ϕ_i^{final} are the cores on which, actors $\boldsymbol{a}_i$ is mapped before and after task migration, respectively and E_{bit} is the energy required to communicate every bit of data across the NoC (refer to Chapter 3 Equation 3.9).

5.3 Design Methodology

The fault-tolerant task mapping methodology consists of two phases – analysis of applications at design-time and execution at run-time. The focus of this work is on the design-time analysis; however, for the sake of completeness, a brief overview is provided on how to use the design-time analysis results at run-time.

The fault-tolerant task mapping methodology is outlined in Figure 5.2. For every fault-scenario with f faulty cores, an optimal mapping is generated that satisfies the throughput requirement and results in minimum energy consumption. These mappings are encoded by the *Encode Mapping* block and stored in memory. At run-time, an application is executed until faults occur. On the detection of a fault[2], the corresponding fault-scenario is identified and the encoded mapping is fetched from the memory. This mapping is then decoded by the *Decode Mapping* block and forwarded to the *Task Migration* block where actual migration is carried out[3].

[2]This work is orthogonal to any fault-detection mechanism

[3]It is to be noted that, mappings and schedules determined at design-time for different fault-scenarios satisfy an application throughput requirement. By enforcing these mappings and schedules at run-time post fault occurrences, throughput is guaranteed for the application under all processor fault-scenarios.

Algorithm 9 Generate fault-tolerant mappings

Input: Initial mapping $M_{n_{arc}}$, G_{app}, G_{arc}, throughput constraint $\mathbb{T}_c$, fault-tolerance level F
Output: Minimum energy mappings for all fault-scenarios with $f = 1$ to F faults

1: **for** $f = 1$ to F **do**
2: $S^f = genFaultScenarios(f)$
3: **for** $s_f \in S^f$ **do**
4: $s_f = (\boldsymbol{c}_{i_1}, \boldsymbol{c}_{i_2}, \cdots, \boldsymbol{c}_{i_{f-1}}, \boldsymbol{c}_{i_f})$ *//represent fault-scenario*
5: $s_{f-1} = (\boldsymbol{c}_{i_1}, \boldsymbol{c}_{i_2}, \cdots, \boldsymbol{c}_{i_{f-1}})$ *//generate reduced fault-scenario*
6: $M_{f-1} = HashMap[s_{f-1}].getMap()$ *//fetch mapping for reduced fault-scenario*
7: $M_f = genMinEnergyMap(M_{f-1}, G_{app}, G_{arc}, \mathbb{T}_c, \boldsymbol{c}_{i_f}, s_f)$ *//generate minimum energy map*
8: $HashMap[s_f].setMap(M_f)$ *//store mapping for the fault-scenario*
9: **end for**
10: **end for**

5.4 Fault-tolerant Mapping Generation

Fault-tolerant mappings are generated using Algorithm 9. There are F stages of the algorithm, where F is a user-defined parameter denoting the maximum number of faults to be tolerated in the device. At every stage f ($1 \leq f \leq F$), mappings are generated – one for each fault-scenario with f faulty cores.

The first step at every stage of the algorithm is the generation of a set (S^f) of fault-scenarios (line 2). The cardinality of this set (denoting the number of fault-scenarios) is ${}^{N_c}P_f$. An example set with 2 out of 3 cores as faulty ($f = 2, N_c = 3$) is the set $S^f = \{\langle 0,1\rangle, \langle 1,0\rangle, \langle 0,2\rangle, \langle 2,0\rangle, \langle 1,2\rangle, \langle 2,1\rangle\}$[4]. For every scenario of the set S^f, the last core ($\boldsymbol{c}_{i_f}$) of the tuple $\langle \boldsymbol{c}_{i_1}, \boldsymbol{c}_{i_2}, \cdots, \boldsymbol{c}_{i_f}\rangle$ is considered as the current faulty core, and a lower order tuple is generated by omitting $\boldsymbol{c}_{i_f}$ (line 5). This gives fault-scenario s_{f-1} with $f-1$ faulty cores for which the optimal mapping is already computed (and stored in $HashMap$) in the previous stage (i.e., at stage $f-1$). As an example, the fault-scenario $\langle 3,1,5\rangle$ implies that faults occurred first on core $\boldsymbol{c}_3$, followed by on core $\boldsymbol{c}_1$, and finally on core $\boldsymbol{c}_5$. Thus, to reach this fault-scenario, the system need to encounter fault-scenario $\langle 3,1\rangle$ first. Mapping for $\langle 3,1\rangle$ is therefore considered as the starting mapping for $\langle 3,1,5\rangle$ with core $\boldsymbol{c}_5$ as current failing core. Similarly, mapping for $\langle 3\rangle$ is the starting mapping for scenario $\langle 3,1\rangle$, with core $\boldsymbol{c}_1$ failing next. A point to note here is that, the scenario $\langle 3\rangle$ is a single fault-scenario, and to reach this, the starting mapping is the no fault initial mapping M_{N_c}.

An important aspect of Algorithm 9 is the generation of the minimum energy mapping $genMinEnergyMap()$. This routine takes a starting mapping M_n, the current faulty core ($\boldsymbol{c}_j$), and the fault-scenario (s_f) and generates a new mapping M_{n-1} with core $\boldsymbol{c}_j$ as faulty. This new mapping satisfies the throughput constraint and gives minimum

[4] A fault-scenario (0,1) implies fault occurring first at core $\boldsymbol{c}_0$ and then at core $\boldsymbol{c}_1$. Thus, fault-scenario (0,1) is different from fault-scenario (1,0) implying a permutation in the fault-scenario computation.

Algorithm 10 $GenMinEnergyMap()$: Energy aware mapping

Input: Mapping M_n, $\mathcal{G}_{app}$, $\mathcal{G}_{arc}$, throughput constraint $\mathbb{T}_c$, faulty core $\boldsymbol{c}_\vartheta$ and fault-scenario s_f
Output: New mapping M_{n-1}
1: Γ_ϑ = Set of mappings generated from M_n by remapping all $\boldsymbol{a}_i \in \Psi(\boldsymbol{c}_\vartheta)$ to some $\boldsymbol{c}_j \in \mathbb{C} \setminus s_f$
2: Sort the mappings in Γ_ϑ according to communication energy and $M_{temp} = \Gamma_\vartheta[0]$
3: Initialize $numIter = 0$; $M_{best} = M_{temp}$; $E_{best} = calcEnergy(M_{temp})$
4: **while** $numIter \leq maxIter$ **do**
5: $[i\ j\ k] = RemapActor(M_{temp}, \mathcal{G}_{app}, \mathcal{G}_{arc}, \mathbb{T}_c, s_f)$
6: **if** $i \geq 0$ **then**
7: $x_{ij} = 1$ and $x_{ij'} = 0\ \forall j' \neq j$; $y_{ik} = 1$ and $y_{ik'} = 0\ \forall k' \neq k$; Update M_{temp}
8: **else**
9: $E = calcEnergy(M_{temp})$
10: **if** $E < E_{best}$ **then**
11: $M_{best} = M_{temp}$; $E_{best} = E$
12: **end if**
13: $numIter++$
14: $M_{temp} = \Gamma_\vartheta[numIter]$
15: **end if**
16: **end while**
17: Return M_{best}

energy (computation and communication). Details of this routine are provided in the next section. Once an optimal mapping is determined (line 7), the algorithm stores it in the $HashMap$ for the particular fault-scenario (line 8). This is repeated for every scenario of set S^f.

5.4.1 Generate Minimum Energy Mapping

Mapping and scheduling applications on a multiprocessor platform is an NP-hard problem. A heuristic is proposed to simplify this process. This is shown as a pseudo-code in Algorithm 10. The algorithm has two sections – remapping the mandatory actors (line 1) and searching for the minimum energy mapping (lines 4 - 16). The mandatory mappings are generated by remapping only the tasks on the faulty core ($\boldsymbol{c}_\vartheta$). These actors are denoted by the set $\Psi(\boldsymbol{c}_\vartheta)$. This is done by selecting $|\Psi(\boldsymbol{c}_\vartheta)|$ cores (not all different) from the set of operating cores $\mathbb{C} \setminus s_f$ to remap all $\boldsymbol{a}_i \in \Psi(\boldsymbol{c}_\vartheta)$. The number of such mappings is equal to the number of ways of choosing a sample of $|\Psi(\boldsymbol{c}_\vartheta)|$ balls with replacement from a set of $|\mathbb{C} \setminus s_f|$ balls. This is equal to $|\mathbb{C} \setminus s_f|^{|\Psi(\boldsymbol{c}_\vartheta)|}$. These mappings are pruned according to standard speed-up techniques (such as processor load [28]). These mappings are stored in an array Γ_ϑ, and the array is sorted in terms of communication energy (line 2). The $maxIter$ best mappings are selected and used in the next stage. This number ($maxIter$) is equal to the number of iterations of the performance section and determines the termination (and hence the execution time) of the algorithm. It is to be noted that the communication-energy-based sorting provides better results (i.e., less energy) than migration-overhead- or throughput-based mappings.

The minimum energy search section of the algorithm (lines 4 - 16) remaps one or

Algorithm 11 *RemapActor*(): Remap actors to minimize energy

Input: Mapping M, $\mathcal{G}_{app}$, $\mathcal{G}_{arc}$, throughput constraint $\mathbb{T}_c$, fault-scenario s_f
Output: Determine an actor to be remapped, the corresponding core and *operating point*

```
1: E := calcEnergy(M); [S  T] = MSDF^3(M, G_app, G_arc, T_c); G_best = 0; i_best = j_best = k_best = −1
2: for all a_i ∈ A do
3:     for all c_j ∈ C \ s_f do
4:         for all k ∈ [0, 1, ··· N_f − 1) do
5:             M_new = M; Set x_ij = y_ik = 1 and x_ij' = y_ik' = 0 ∀j' ≠ j and k' ≠ k
6:             [S_new  T_new] = MSDF^3(M_new, G_app, G_arc, T_c); E = calcEnergy(M_new)
7:             if ((T_new ≥ T_c) && (E_new < E)) then
8:                 G = (E − E_new)/(T_new − T)
9:                 if G > G_best then
10:                    G_best = G; i_best = i; j_best = j; k_best = k
11:                end if
12:            end if
13:        end for
14:    end for
15: end for
16: return [i_best j_best k_best]
```

more actors selectively to determine the minimum energy. At each iteration, the starting mapping is one of the mappings of set Γ_ϑ. The *RemapActor*() routine selects an actor to be remapped, satisfying the throughput requirement. If the return set is nonempty (implying actors can be remapped without violating the throughput constraint), the actor is remapped to a core at an *operating point* determined by the *RemapActor*() routine (line 7). The process is continued as long as no actors can be found to be remapped without violating the throughput. When this happens (lines 8 - 15), the total energy of the mapping is calculated using the *calcEnergy*() routine that incorporates (1) the computation energy; (2) the communication energy; and (3) the migration energy.

If the total energy of the mapping is lower than the minimum energy (E_{best}) obtained thus far, the best values are updated (line 11). The number of iterations is incremented (line 13) and the whole search is repeated starting from the next mapping in the set Γ_ϑ. The algorithm terminates when *numIter* becomes equal to *maxIter* and the best mapping is returned (line 17).

5.4.2 Minimum Energy Actor Remapping

Algorithm 11 provides the pseudo-code for the *RemapActor*() subroutine that uses a gradient function to evaluate each actor to core assignment. The total energy and the throughput are evaluated by assigning every actor to every core at every operating point (line 6). The $MSDF^3$ tool (refer to Chapter 3) is used to compute the schedule and throughput from a given mapping[5]. If the throughput for the new assignment is greater than the throughput constraint and the energy is lower than the energy of the initial

[5]For DAGs, multiple iterations are usually executed sequentially (in a non-overlapped manner). For these graphs *CPTO* routine of [149] can be used to compute the performance measured as *makespan*.

Algorithm 12 Generate initial mapping

Input: $\mathcal{G}_{app}$, $\mathcal{G}_{arc}$ and throughput constraint $\mathbb{T}_c$
Output: Minimum energy initial mapping M
1: Initialize $[M \;\; \mathcal{S} \;\; \mathbb{T}] = SDF^3(\mathcal{G}_{app}, \mathcal{G}_{arc}, \mathbb{T}_c)$
2: **while** true **do**
3: $[i \;\; j \;\; k] = RemapActor(M, \mathcal{G}_{app}, \mathcal{G}_{arc}, \mathbb{T}_c, \emptyset)$
4: **if** $i \geq 0$ **then**
5: Update M with $x_{ij} = y_{ik} = 1$ and $x_{ij'} = y_{ik'} = 0 \; \forall j' \neq j$ and $k' \neq k$
6: **else**
7: break
8: **end if**
9: **end while**
10: Return M

mapping M, the gradient is computed (line 8). If the gradient is higher than the best gradient obtained thus far, the best values are updated (line 11). The best actor, core, and operating point are returned.

5.4.3 Generate Initial Mapping

Algorithm 12 provides the pseudo-code for the initial mapping generation procedure of the proposed methodology. The initial mapping (at line 1) is obtained by any deterministic task mapping and scheduling algorithm e.g., $HEFT$ of [150] for DAGs and the unmodified SDF^3 tool for SDFGs. The $RemapActor()$ routine selects one actor to be remapped to a core at a frequency such that energy is minimized with least degradation of throughput. This follows the same principle as that of Algorithm 10 with the all working cores i.e., setting $s_f = \emptyset$.

5.5 Fault-tolerant Scheduling

An important aspect of any application graph (cyclic and acyclic) is the scheduling of actors on cores. There are different scheduling schemes proposed, both for DAGs and SDFGs [80,151]. None of the existing fault-tolerant techniques address scheduling. If the run-time schedule is different from that used for analysis at design-time, the throughput obtained will be significantly different than what is guaranteed at design-time. There are therefore two approaches to solve the problem:

- store the actor mapping and scheduling for all fault-scenarios and for all applications from design-time (*storage-based*); and
- constructs the schedule at run-time based on the mappings stored from the design-time (*construction-based*).

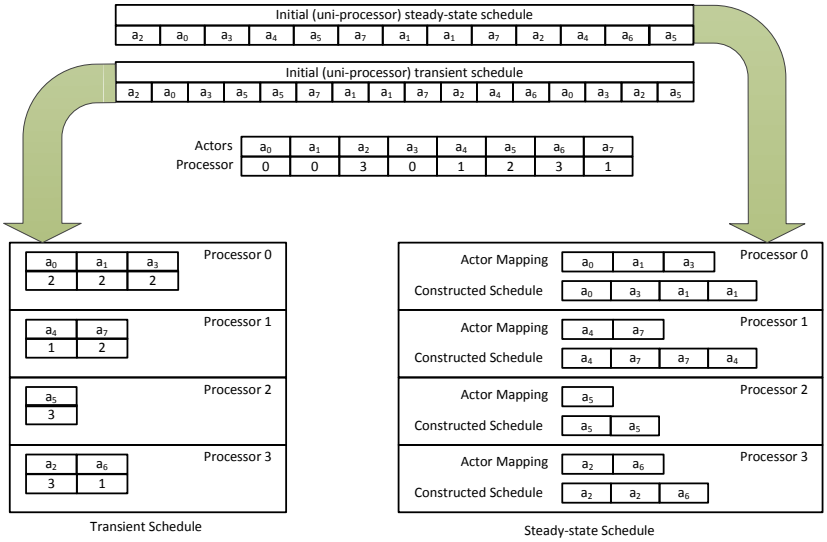


Figure 5.3: Schedule construction from an initial schedule and actor allocation.

The former is associated with high storage overhead and the latter with longer execution time. Both storage and execution time overhead are crucial for streaming applications. A self-timed execution based scheduling is proposed to solve the two problems.

Based on the basic properties of self-timed scheduling, it can be proven that if the schedule of actors on a uniprocessor system is used to derive the schedules for a multiprocessor system maintaining the actor firing order, the resultant multiprocessor schedule would be free of deadlocks [152]. However, the throughput obtained using this technique could be lower than the maximum throughput of a multiprocessor schedule constructed independently. Thus, as long as this throughput deviation is bounded, the schedule for any processor could be easily constructed from the mapping of actors to this processor and a given uniprocessor schedule.

Figure 5.3 shows the operation of the proposed scheduling technique. The actor-core mapping indicates that actors $\boldsymbol{a}_0$, $\boldsymbol{a}_1$ and $\boldsymbol{a}_3$ are mapped to core 0. The initial steady-state schedule indicates that there are two instances of $\boldsymbol{a}_1$ and one each for actors $\boldsymbol{a}_0$ and $\boldsymbol{a}_3$, respectively. The steady-state order of actor firing on core 0 is determined from this initial schedule by retaining only the mapped actors. In a similar way, the steady-state schedules are constructed for all other processors. The transient part of the schedules are constructed from the given initial uniprocessor transient schedule by retaining the mapped actors. However, the only difference of the transient-phase schedule construction with the steady-state phase is that for the transient phase, the number of actors firing is important and not the exact order. This is indicated by a number against each actor for

Algorithm 13 Schedule generation

Input: $\mathcal{G}_{app}$, $\mathcal{G}_{arc}$, $\mathbb{T}_c$, N_s and Δ
Output: Schedule for all fault-scenarios

```
1: forall f ∈ [1 ··· F] do S^f = genFaultScenarios(f)
2: maxIter = |S^f|;  sDB = constructUniSchedule(G_app, N_s);  sDB_t = sDB
3: while S^f ≠ ∅ do
4:     for all schedule l_i ∈ sDB do
5:         Initialize count = 0
6:         for all s_f ∈ S^f do
7:             M_temp = HashMap[s_f].getMap();  T = SMSDF^3(M_temp, l_i, G_app, G_arc)
8:             if T ≥ T_c then
9:                 count++
10:            end if
11:        end for
12:        l_i.rank = count
13:    end for
14:    l_min = getHighestRankSchedule(sDB)
15:    for all s_f ∈ S^f do
16:        M_temp = HashMap[s_f].getMap();  T = SMSDF^3(M_temp, l_min, G_app, G_arc)
17:        if T ≥ T_c then
18:            Sche[s_f] = l_min;  S^f.eliminate(s_f)
19:        end if
20:    end for
21:    numIter++
22:    if numIter > maxIter then
23:        numIter = 0;  C = C − Δ
24:    end if
25: end while
```

each processor, as shown in the figure.

During the steady-state operation, every core maintains counts of the number of remaining steady-state firings for the actors mapped to the core. These numbers are updated when an actor completes its execution. When a fault occurs, the mapped actors on the faulty core are moved to new location(s) (cores), along with the remaining firing count. On such cores, which have at least one incoming migrated actor, all actors are allowed to execute in a self-timed manner to finish the remaining firing counts of the current pending iteration (similar to the initial transient phase). From the subsequent iteration onwards, the steady-state order can be enforced for the moved actors. This will prevent the application from going into deadlock when a fault occurs. In determining actor counting in the steady-state iterations, schedule minimization is disabled. As an example, in Figure 5.3, the steady state schedule constructed for core 2 consists of two executions of actor $\boldsymbol{a}_5$ as opposed to one in the otherwise minimized schedule.

Algorithm 13 provides the pseudo-code for the modified self-timed execution technique for generating the steady-state schedule. The first step towards this is the construction of uni-processor schedules (line 2). A *list scheduling* technique is adopted for this purpose along with several algorithms for tie-breaking, for example, ETF (earliest task first), DLS (dynamic level scheduling), etc. These algorithms are implemented in the $constructUniSchedule()$ routine. The number of uni-processor schedules constructed us-

ing this routine is a user-defined parameter N_s. These schedules are stored in a database in memory (sDB). The list of fault-scenarios possible with F faults are also listed in the set S^f. Using each uni-processor schedule as the initial schedule, throughput is computed for the given application for all fault-scenario mappings. The $SMSDF^3$ computes the throughput of a mapping using a given uni-processor schedule.

For each uni-processor schedule from sDB, a count (termed as $rank$) is determined (lines 4 - 13). The value indicates the number of fault-scenarios for which the throughput constraint is satisfied with this as the initial schedule. The schedule with the highest rank is selected and assigned as the initial schedule for the successful fault-scenarios (lines 14 - 20). A fault-scenario is termed successful with respect to a schedule if the throughput constraint is satisfied with the given schedule. The successful fault-scenarios are discarded from the list of fault-scenarios (S^f). The process is repeated as long as the set S^f is non-empty. The limited set of uni-processor schedules does not guarantee throughput satisfiability for all fault-scenarios. If such a fault-scenario exists, S^f is never $\emptyset$ causing the algorithm to be stuck in a loop. To avoid such situations, a check is performed (line 22 - 24) to limit the number of iterations. The maximum number of iterations is upper bounded by the number of fault-scenarios. Every time the iteration count reaches this value, the throughput constraint is decremented by a small quantity, Δ. The algorithm thus allows for a graceful performance degradation. The granularity of this is based on the execution time and solution quality trade-off.

5.6 Results

5.6.1 Experimental Setup

Experiments are conducted on synthetic and real application graphs on Intel Xeon 2.4 GHz server running Linux. Fifty synthetic applications are generated with the number of actors in each application selected randomly from the range 8 to 25. Additionally, fifteen real applications are considered with seven from streaming and the remaining eight from non-streaming domain. The streaming applications are obtained from the benchmarks provided in the SDF^3 tool [138]. These are *H.263 Encoder*, *H.263 Decoder*, *H.264 Encoder*, *MP3 Decoder*, *MPEG4 Decoder*, *JPEG Decoder* and *Sample Rate Converter*. The non-streaming application graphs considered are *FFT*, *Romberg Integration* and

VOPD from [142] and one application each from *automotive*, *consumer*, *networking*, *telecom* and *office automation* benchmark suite [153]. These applications are executed on a multiprocessor system consisting of 4 to 16 cores arranged in a mesh-based topology. A heterogeneity of 3 ($h = 3$) is assumed for the cores i.e. each core can be of one of the three different types. Four *operating points* are assumed for each core.

All algorithms developed in this work are coded in *C++*. Since this is the first work on *reactive* fault-tolerance considering throughput, computation and communication energy optimization jointly, there are no existing works for comparison. However, results of this work are compared with some of the existing *reactive* fault-tolerant techniques such as the throughput maximization technique of [120] (referred to as *TMax*), the migration overhead minimization technique of [118] (referred as *OMin*), the energy minimization technique of [41] (referred as *EMin*), the throughput constrained migration overhead minimization technique of [75] (referred as *TConOMin*) and the throughput constrained communication energy minimization technique of [74] (referred as *TConCMin*). The technique proposed here minimizes total energy (computation and communication energy) with throughput as a constraint and is referred as *TConEMin*. The objective of these comparisons is to establish the fact that the existing techniques when applied to multiprocessor systems can lead to sub-optimal results in terms of energy consumption and throughput per unit energy metric.

5.6.2 Complexity Analysis of Algorithms

There are three algorithms proposed in this work – fault-tolerant mapping generation algorithm (Algorithms 9, 10 and 11), the initial mapping generation algorithm (Algorithm 12) and the schedule generation algorithm (Algorithm 13). The complexity of Algorithm 9 is calculated as follows. The number of iterations of the algorithm is determined by the number of fault-scenarios with F faults. This is given by Equation 5.5.

$$N_{FS} = \sum_{f=1}^{F} {}^{N_c}P_f \tag{5.5}$$

At each iteration, the $genMinEnergyMap$ algorithm is invoked. The overall complexity of Algorithm 9 is given by Equation 5.6 where C_{10} is the complexity of Algorithm 10.

$$O(C_9) = O\left(N_{FS} \cdot O(genMinEnergyMap)\right) = O(N_{FS} \cdot C_{10}) \tag{5.6}$$

The complexity of Algorithm 10 is governed by two factors – parameter $maxIter$ and the routine $RemapActor()$. Core and frequency assignments for an actors are accomplished in constant time. Assuming the $RemapActor()$ routine to be executed η times on average for each value of $numIter$, the complexity of Algorithm 10 is

$$C_{10} = maxIter \cdot \eta \cdot O(RemapActor) \tag{5.7}$$

The $RemapActor()$ routine remaps each actor on each functional core at each frequency to determine if the throughput constraint is satisfied and the energy is lower than the minimum energy obtained thus far. If actor assignment operations take unit time and the complexity of the $MSDF^3$ engine is denoted by $O(MSDF^3)$, the overall complexity of Algorithm 11 is given by Equation 5.8.

$$O(RemapTask) = C_{11} = O(N_a \cdot N_c \cdot N_o \cdot O(MSDF^3)) \tag{5.8}$$

Combining Equations 5.6, 5.7 and 5.8, the complexity of the fault-tolerant mapping generation algorithm is given by equation 5.9.

$$C_9 = O(N_{FS} \cdot maxIter \cdot \eta \cdot N_a \cdot N_c \cdot N_o \cdot O(MSDF^3)) = O\left(N_a^{F+4} \cdot N_o\right) \tag{5.9}$$

where $N_c \leq N_a$, N_{FS} can be upper bounded by N_c^F (in big O notation) and $O(MSDF^3) = O\left(N_a \log N_a + N_a \cdot \pounds\right)$, where $\pounds(\leq N_a)$ is the average number of successors of an actor.

The complexity of the schedule generation algorithm (Algorithm 13) is calculated as follows. The rank computation for all the uni-processor schedules can be performed in $O(N_s \cdot N_{FS})$ time, where N_s is the number of uni-processor schedules constructed and N_{FS} is the number of fault-scenarios. The highest throughput rank can be selected in $O(N_s)$ and lines 15 - 20 can be performed in $O(N_{FS} \cdot O(SMSDF^3))$. Finally, the outer while loop (lines 3 - 25) is repeated N_{FS} times in the worst case. Combining,

$$C_{13} = O\left(N_{FS} \cdot (N_s \cdot N_{FS} + N_s + N_{FS} \cdot O(MSDF^3))\right) = O\left((N_s + O(MSDF^3)) \cdot N_a^{2F}\right) \tag{5.10}$$

5.6.3 Selection of Initial Mapping

Figure 5.4 plots the throughput and the energy performance of the proposed technique in comparison with the three prior research works on *reactive* fault-tolerance. The starting mapping selection criteria for these works are highest throughput for *TMax* [120], migration overhead minimization with throughput constraint for *TConOMin* [75]), and communication energy minimization with throughput constraint for *TConCMin* [74]),

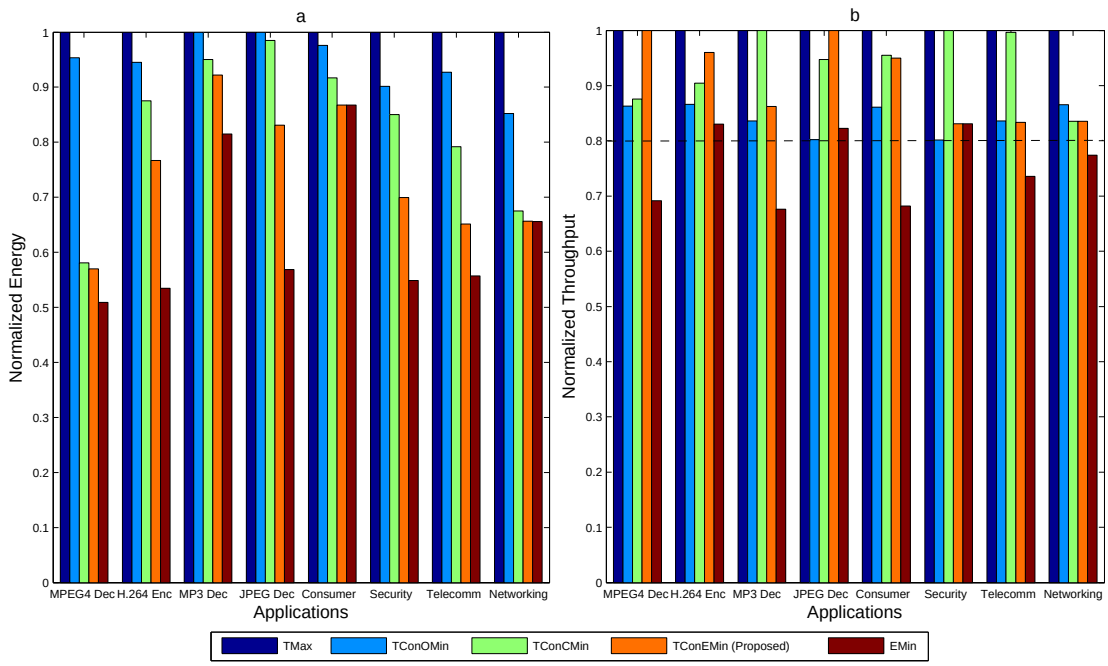


Figure 5.4: Energy and performance for different applications.

respectively. Additionally, to determine the energy overhead incurred in considering throughput in the optimization process, the proposed technique is also compared with the minimum energy starting mapping (*EMin* of [41]).

Figure 5.4(a) plots the normalized total energy consumption per iteration of 8 real-life applications for the existing and the proposed techniques. The energy values are normalized with respect to those obtained using *TMax*. As can be seen from the figure, the energy consumption of the proposed technique (*TConEMin*) is the least among all the existing *reactive* fault-tolerant techniques. This trend is also true for all the 42 remaining applications considered (not shown explicitly here). On average, for all the applications, *TConEMin* achieves 30%, 25% and 16% less energy as compared to the *TMax, TConOMin* and *TConCMin*, respectively. The energy savings with respect to *TConCMin* is lower as compared to the other two techniques, because *TConCMin* minimizes communication energy component of the total energy, while the other two techniques do not consider energy optimization. Finally, the *TConEMin* consumes 15% more energy than *EMin*, which does not consider throughput degradation.

Figure 5.4(b) plots the normalized throughput of all the techniques. The throughput constraint is shown by the dashed line in the figure. As previously indicated, the *EMin* does not consider throughput degradation, and therefore, throughput constraint is violated for most applications (a total of 45 out of all 50 applications).

Table 5.1: Number of mappings in exhaustive search.

Actors	Homogeneous	Heterogeneous	
	1 core type	2 core types	3 core types
2	2	6	12
4	15	94	309
6	203	2,430	12,351
8	4,140	89,918	681,870
10	115,975	4,412,798	48,718,569
14	190,899,322	20,732,504,062	461,101,962,108

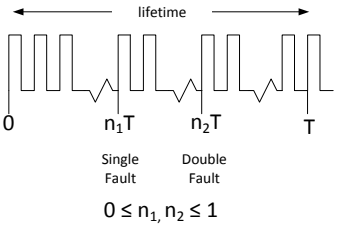


Figure 5.5: Simulation environment.

Another aspect of the starting mapping generation algorithm is the execution time. The reactive fault-tolerant techniques in [74, 75, 120] search the design space exhaustively to select a starting mapping. Although this is solvable for homogeneous cores with a limited number of actors and/or cores, the same becomes computationally infeasible, even for small problem size, as the cores become heterogeneous. Table 5.1 reports the growth in the size of the design-space (number of mappings evaluated) as the number of actors scales. The number of cores in the table is same as the number of actors. If the SDF^3 engine takes an average $10\mu S$ to compute the schedule of a mapping, the design-space exploration time for 14 actors on 14 cores with three types of heterogeneous cores is 54 days. The proposed heuristic solves the same problem in less than two hours.

5.6.4 Energy Savings With Core fault-scenarios

This section introduces the energy savings obtained during the overall lifetime of an MPSoC as one or more permanent faults occur. Experiments are conducted with the same set of applications (50 in total) and executed on an architecture with 2×3 cores. The number of faults is restricted to 2. These are forced to occur after $n_1 \cdot T$ and $n_2 \cdot T$ years, respectively from the start of the device operation, where T is the total lifetime of the device and $0 \leq n_1, n_2 \leq 1$. Figure 5.5 represents the simulation environment. During 0 to $n_1 \cdot T$ years, energy is consumed by the starting mapping, that is, the no-fault mapping obtained in Subsection 5.6.3; during $n_1 \cdot T$ years to $n_2 \cdot T$ years and $n_2 \cdot T$ years to T years, energy is consumed by single fault-tolerant and the double fault-tolerant

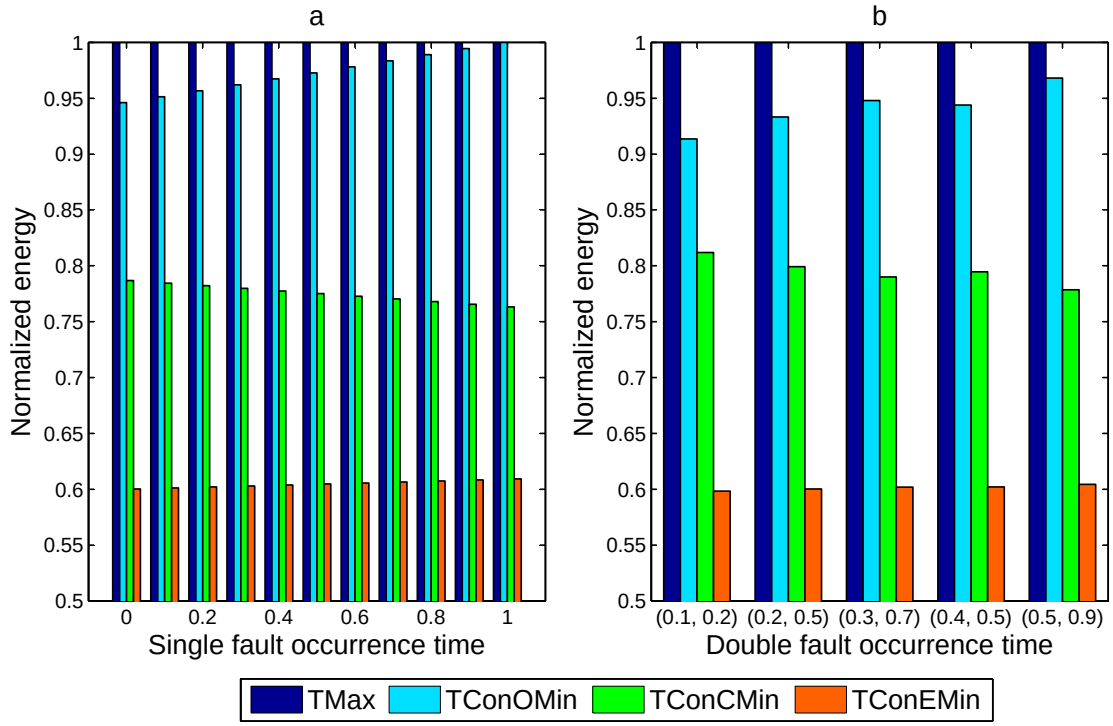


Figure 5.6: Lifetime energy consumption with single and double faults.

mappings, respectively. The cores affected by faults are selected randomly, and the results presented here are an average of all single and double faults for all applications.

Figure 5.6(a) plots the result for single-fault scenario, that is, assuming only single fault occurs during the lifetime of the device. The average energy per iteration of the application is plotted with n_1 varied from 0 to 1. A lower value of n_1 implies a fault occurs in the early life of the device, while a higher value indicates faults occurring at later stages. Since *EMin* does not consider fault-scenarios, only *reactive* fault-tolerant techniques (with throughput consideration) are included for comparison.

As can be seen from this figure, the energy consumption of *TMax* and *TConOMin* techniques are comparable and is higher than that consumed by the other two techniques. This is due to the non-consideration of computation and communication energy for optimization. Although *TConOMin* minimizes migration overhead (energy), this energy is one-time overhead (i.e., incurred during fault) and is negligible compared to the total energy consumed in the lifetime of the device. *TConCMin* considers communication energy and throughput jointly, and therefore the energy is lower than *TMax* and *TConOMin* by average 23% and 20%, respectively. The proposed *TConEMin* minimizes the total energy by achieving an average 22% savings as compared to *TConCMin*.

Figure 5.6(b) plots the result for double-fault scenarios. A fault-coordinate (n_1,n_2)

Table 5.2: Execution time (in secs) of existing and proposed technique.

Actors	Homogeneous (1 core type)				Heterogeneous (3 core types)			
	Existing		Proposed		Existing		Proposed	
	4 cores	9 cores	4 cores	9 cores	4 cores	9 cores	4 cores	9 cores
4	110	1,450	100	1,210	150	2,150	150	1,900
8	630	6,770	410	3,100	1,810	17,440	720	5,980
12	80,100	–	1,320	6,600	2,47,000	–	2,280	12,700
16	–	–	9,700	10,600	–	–	16,750	22,400

refers to the time for the first and the second fault occurrence, respectively. Although, experiments are conducted for all values of the fault-coordinates, results for a few of the coordinates are plotted. Similar to the single fault results, the proposed *TConEMin* also achieves 30% lower energy as compared to the existing techniques for multiprocessor system with two faults. These results prove that energy-aware mapping selection for fault-tolerance is crucial for minimizing the total energy consumption of a system.

5.6.5 Execution Time Comparison of the Proposed Mapping Algorithm

As established previously, all prior fault-tolerant works search for a suitable mapping exhaustively for different fault-scenarios. A dynamic programming is proposed in [120] to compute the minimum migration overhead incurred in moving from an initial mapping to the fault-scenario mapping. An ILP approach is proposed as an alternative in [74, 75] to compute the minimum migration overhead. However, selection of the fault-tolerant mapping is based on exhaustive search. Although dynamic programming and ILP are computationally feasible for small problem sizes, the bottleneck is in the exhaustive mapping selection process (which grows exponentially with the number of actors and cores), limiting their adaptability for large problem sizes and heterogeneous architectures. This work addresses this problem by proposing a heuristic algorithm with worst-case complexity, given by Equation 5.9. Table 5.2 reports the execution time of the existing approaches in comparison with the proposed heuristic for homogeneous and heterogeneous architectures with different actors and core count. For the existing approaches, the execution time reported in the table includes mapping generation time, mapping evaluation time (throughput computation), and dynamic programming (or ILP) time for fault-tolerant mapping selection. For the proposed approach, the execution times is the sum of the execution times of Algorithms 9 and 12.

There are a few trends to be followed from this table. First, the execution time for the heterogeneous architecture is longer than that for the homogeneous architecture for all

Table 5.3: Iterations with the proposed and simulated annealing based heuristic.

Actors	Proposed Technique						Simulated Annealing		Speedup	
	4 cores			9 cores			4 cores	9 cores	4 cores	9 cores
	maxIter	η	Total	*maxIter*	η	Total				
4	100	5	500	100	5	500	2,100	3,900	4.2×	7.8×
8	100	6	600	100	8	800	4,400	6,100	7.3×	7.6×
12	100	8	800	100	12	1,200	7,300	12,700	9.2×	10.6×
16	100	10	1,000	100	14	1,400	11,700	21,400	11.7×	15.3×

Table 5.4: Migration overhead performance.

		Migration Energy (nJ)	Total Energy (nJ)	Migration Overhead Savings (nJ)	Extra Energy Per Iteration (nJ)	Iterations to recover
H.264 Encoder	*OMin*	1.1×10^9	7.2×10^5	7×10^8	3.2×10^5	2,188
	TConOMin	1.7×10^9	4.6×10^5	1×10^8	6×10^4	1,667
	TConEMin	1.8×10^9	4.0×10^5	–	–	–
MP3 Decoder	*OMin*	7.0×10^8	2.9×10^6	1.7×10^9	1.4×10^6	1,215
	TConOMin	1.3×10^9	2.0×10^6	1.1×10^9	5×10^5	2,200
	TConEMin	2.4×10^9	1.5×10^6	–	–	–

actor-core combinations. This trend is the same for the proposed approach as well as the existing approaches. This is due to the fact that with core heterogeneity, the execution time of an actor is different on different cores, and therefore more actor-core combinations are evaluated. Second, the execution time of the proposed approach is comparable with that of the existing approaches for fewer actors (four in the table) due to the fewer number of exhaustive mappings. Third, as the number of actors increases, the number of actor-core combinations (mappings) grows exponentially, leading to an exponential growth in the execution time for the existing approaches. Beyond 12 actors, these techniques fail to provide a solution due to the high memory requirement (to store the mappings) of the host CPU. The proposed technique scales well with the number of actors and cores. For 12 actors mapped on four cores of three different types, the proposed technique results in 100x reduction in execution time with less than 10% variation from the optimal solutions obtained by solving the Equations 3.5 and 3.10 directly while satisfying the application throughput requirement. Further, Table 5.3 reports the number of iterations performed by the proposed algorithm in comparison with the simulated annealing-based heuristic for different actor and core combinations. The maximum iterations for the proposed algorithm is fixed at 100. This gives a satisfactory result quality (less than 1% deviation from the simulated annealing result). As seen from this table, the proposed approach achieves up to 15x reduction in the number of iterations and hence a correspondingly lower execution time as compared to the simulated annealing-based heuristic.

5.6.6 Migration Overhead Performance

Table 5.4 reports the migration overhead (measured as energy) and total energy of two existing techniques (*OMin* and *TConOMin*) in comparison with the proposed technique for two different applications (*H.264 Encoder* and *MP3 Decoder*) with 5 and 14 actors on a multiprocessor system with 6 cores arranged in 2×3. The core heterogeneity is fixed to 2. The other existing techniques (*TMax, EMin* and *TConCMin*) are not included for comparison, as they do not optimize migration overhead. Columns 3 and 4 report the migration overhead incurred when faults occur and the average energy consumption per iteration of the application graph, respectively. These numbers are average of single- and double-faults values. Column 5 reports the savings in migration overhead achieved by *OMin* and *TConOMin* with respect to the proposed *TConEMin*. Column 5 reports the extra energy (computation + communication) incurred in selecting the same two techniques with respect to *TConEMin*.

As can be seen from the table, significant savings in migration overhead are possible with *OMin* technique. However, this technique is associated with an energy penalty (Column 6). For application *H.264 Encoder*, for example, the migration overhead savings in *OMin* is $7 \times 10^8 nJ$, while the energy penalty is $3.2 \times 10^5 nJ$ per iteration. As established previously, migration is one-time overhead and energy is consumed in every iteration of the application graph (both pre- and post-fault occurrence). Therefore, the savings in migration overhead is compensated in $\frac{7 \times 10^8}{3.2 \times 10^5} = 2,188$ iterations ($\approx 146s$ with a 500MHz clock at encoding rate of 15 frames per sec). This is shown in column 7 of the table. Interpreting this in reverse manner, selecting *TConEMin* as the fault-tolerant technique results in an extra migration overhead of $7 \times 10^8 nJ$, which is amortized in the next (post-fault) $2,188$ iterations of the application graph.

For most of the multimedia applications, actors are executed periodically. Examples of these applications on a mobile phone include decoding of frames while playing video and fetching emails from server. Typically, these applications are executed countably infinite times in the entire lifetime of the device. If N denotes the total iterations of a device post-fault occurrence, then the first $2,188$ iterations will be used to recover the migration overhead loss, while the remaining $(N - 2188)$ iterations will fetch energy savings ($3.2 \times 10^5 nJ$ per iteration). As $N \rightarrow \infty$, the energy savings obtained $= (N - 2188) \times 3.2 \times 10^5 \approx N \times 3.2 \times 10^5 nJ$. This substantial energy gain clearly justifies the

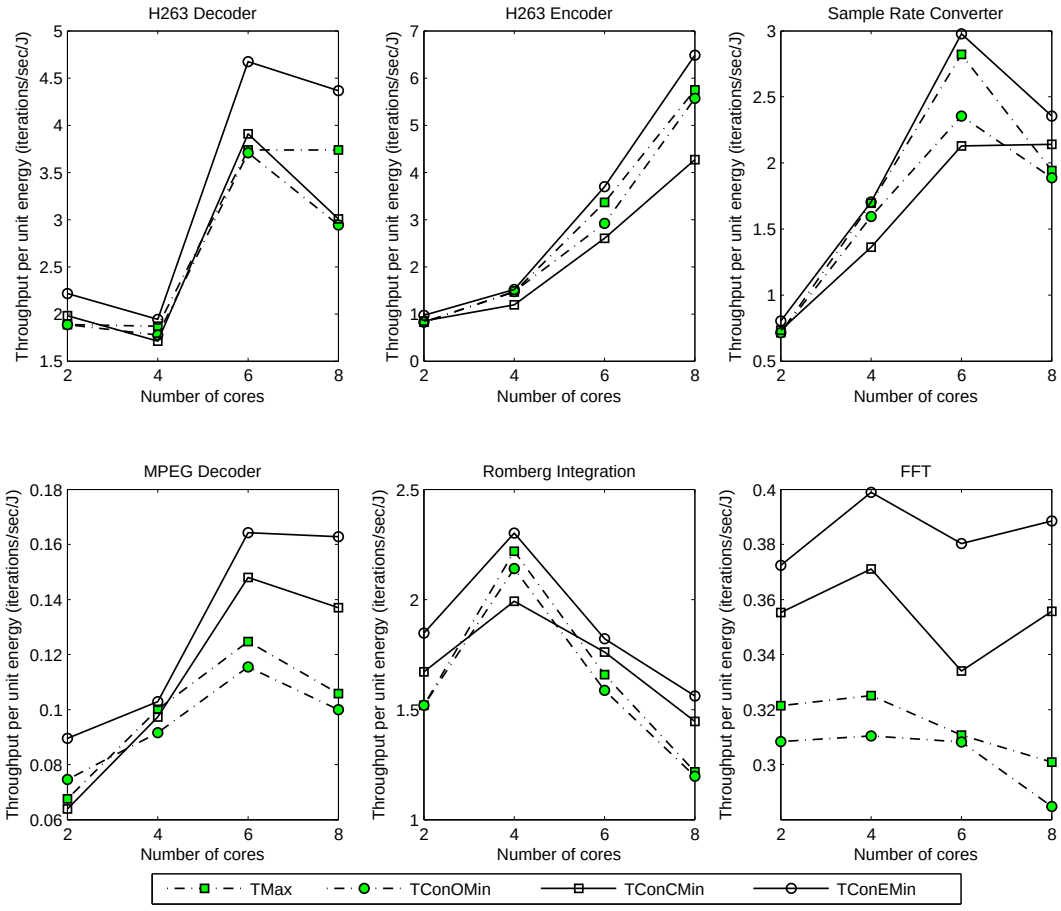


Figure 5.7: Throughput-energy joint performance for real-life applications.

non-consideration of migration overhead in the fault-tolerant mapping selection.

5.6.7 Scalable Throughput Performance

Streaming multimedia applications can be broadly classified into two categories – those benefiting from scalable QoS and those requiring a fixed throughput. Majority of the streaming applications, such as video encoding/decoding, falls in the latter category. The results of the previous sections are based on performance (throughput) as constraint. However, to signify the importance of the proposed technique for scalable throughput applications, a metric is defined (*throughput per unit energy*). The proposed and the existing techniques are compared based on this metric (Figure 5.7). Experiments are conducted with a set of six real applications on an architecture with the number of cores varying from two to eight. Core heterogeneity of the architectures is limited to two as the existing techniques fail to provide a solution for the applications with higher core heterogeneity. The results reported in the figure are the average of all single- and double-fault scenarios. A common trend from these plots is that for most applications (except

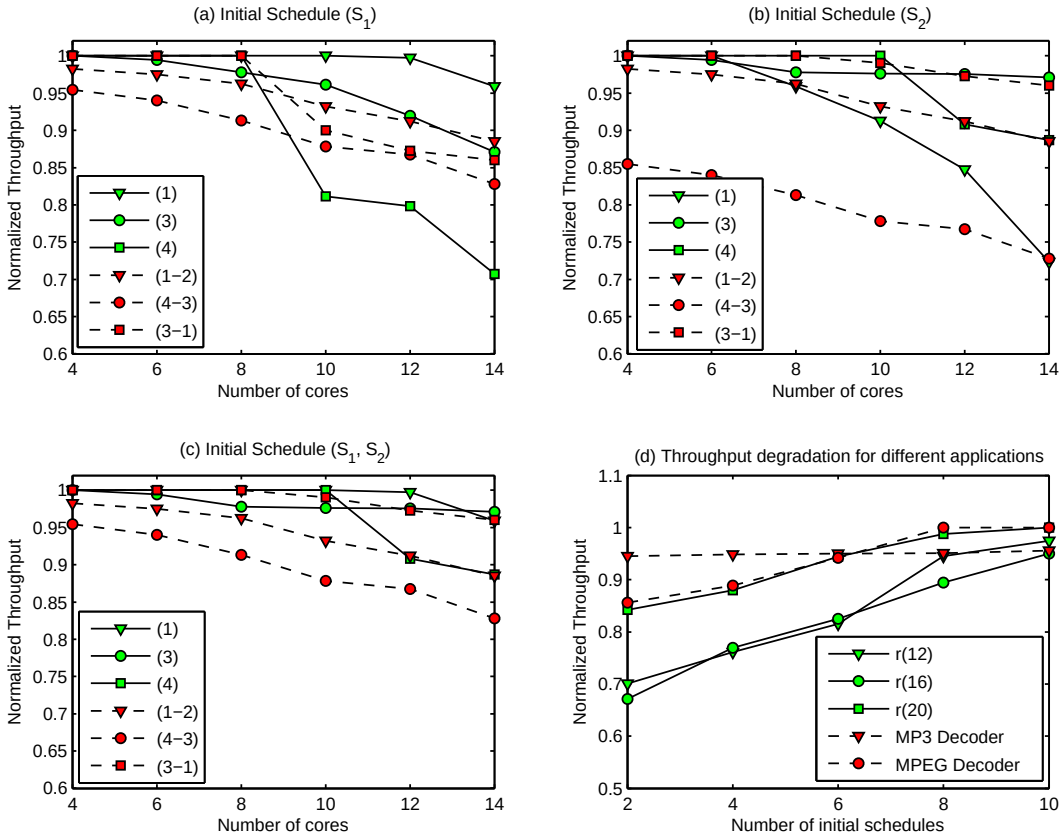


Figure 5.8: Normalized throughput using the proposed self-timed execution.

H.263 Encoder), the throughput per unit energy initially increases with the number of cores. However, beyond a certain core count, the throughput per unit energy decreases. This behavior is the same for all the techniques, that is, *TMax, TConOMin, TConCMin* and *TConEMin*. As the number of cores increases, the throughput of an application increases. At the same time, the two energy components (computation and communication) also increase. For lower core count, the growth in throughput dominates, causing an increase in the overall throughput per unit energy. As the core count increases beyond six cores (four cores for *Romberg Integration* and *FFT*), the energy growth dominates over throughput growth, and therefore, the throughput per unit energy drops. Although H.263 Encoder shows a growth in throughput per unit energy up to 8 cores, the drop-off point is observed for *TConEMin* with 16 cores. However, the results are omitted, as the exhaustive search based existing techniques – *TMax, TConOMin* and *TConCMin* fail to give a solution for that value of core count.

As can be seen, the throughput per unit energy of *TConEMin* is the highest among all existing techniques, delivering on average 30% better throughput per unit energy.

5.6.8 Throughput Performance of the Proposed Scheduling

Figures 5.8a - 5.8c plot the throughput obtained in the proposed self-timed execution-based scheduling technique for six fault-scenarios (three single and three double) of application *MP3 Decoder* as the number of cores is varied from 4 to 14. There are two initial uniprocessor schedules considered ($N_s = 2$). The multiprocessor throughput obtained using these uni-processor schedules are normalized with respect to the throughput obtained using the SDF^3 tool and are plotted in Figures 5.8a and 5.8b.

As can be seen from Figure 5.8a (with initial schedule as S_1), for all fault-scenarios, the normalized throughput decreases with an increase in the number of cores. This is expected, as uniprocessor schedules fail to capture the parallelism available with multiple cores. Among the six fault-scenarios considered, the throughput degradation for fault-scenario *(4)* is the maximum ($\approx 30\%$), while for others, this is less than 20%. Similarly, for Figure 5.8b (corresponding to initial schedule S_2), fault-scenarios *(1)* and *(4-3)* suffer the maximum throughput degradation of 25%. If the two schedules (S_1 and S_2) are considered to be available simultaneously and the one which gives the highest throughput for a fault-scenario is selected as the initial schedule, the throughput degradation can be bounded (predicted) at design-time. This is shown in Figure 5.8c, where S_1 is selected as the initial schedule for fault-scenarios *(1) and (4-3)* and S_2 as the initial schedule for the remaining fault-scenarios. The maximum throughput degradation obtained using this technique is 18%. Figure 5.8d plots the throughput degradation obtained as the number of initial schedules is increased from two to ten for five different applications. The results reported in this plot are the average of all single- and double-fault-scenarios. As can be seen from this figure, the throughput degradation decreases with an increase in the number of initial schedules. On average, for all five applications considered, the throughput degradation is within 5% from the throughput constructed using SDF^3 with 10 initial schedules. A point to note here is that choosing more initial schedules results in an increase in the storage complexity. Results indicate that $N_s = 10$ (i.e 10 initial schedules) offers the best trade-off with respect to storage and throughput degradation.

5.6.9 Scheduling Overhead

Table 5.5 reports the schedule storage overhead (in Kb) and the schedule computation time using the proposed self-timed execution technique in comparison with the storage-

Table 5.5: Schedule storage overhead and computation time.

Parameters	*H.263 Encoder*			*MP3 Decoder*		
	Storage based	Construction based	Proposed	Storage based	Construction based	Proposed
Mapping and schedule storage overhead (Kb)	892.1	68.6	68.6	1464	91.5	91.5
Run-time schedule construction time (s)	0	0.42	0.027	0	3.06	0.035
Design-time schedule construction time (s)	34.44	34.44	2.75	80	80	3.66

based and the construction-based techniques for two applications (*H.263 Encoder* and *MP3 Decoder*) with 5 and 14 actors on an architecture with 12 cores arranged in 3×4. The results are reported for three-fault tolerant systems. The *construction-based* and the proposed technique require storing the fault-tolerant mappings only, while the *storage-based* technique stores the schedule of actors on all cores and for all fault-scenarios alongside the fault-tolerant mappings.

The run-time storage construction overhead is 0 for the *storage-based* technique because schedule needs to be fetched from a database[6]. The *construction-based* technique results in a execution time of 0.4s. The construction time increases exponentially with the number of actors and/or cores (Column 3 and 6). This large schedule-construction time could potentially lead to deadline violations. The proposed technique results in a linear growth of execution time and is scalable with the number of actors and cores.

Finally, the reported execution-time of the design-time analysis phase for the storage-based and the construction-based techniques involves construction of the schedule for all fault-scenarios. Although schedules are not stored in the construction-based technique, they are still computed at design time for verification. The corresponding number for the proposed technique denotes the time for constructing initial schedules only. On average, for all 50 applications considered, the proposed technique reduces storage overhead by 10x (92%) with respect to the *storage-based* technique and execution time by 20x (95%) as compared to the *construction-based* technique.

5.7 Remarks

This work presented a design-time technique to generate mappings of an application on an architecture for all possible core fault-scenarios. The technique minimizes the energy consumption while satisfying the application throughput requirement. Experiments conducted with real and synthetic application graphs on a heterogeneous multiproces-

[6]Fetching of a schedule from a database is faster.

sor platform with different core counts clearly demonstrate that the proposed technique is able to minimize the energy consumption by 22%. Additionally, the technique also achieves 30% better throughput per unit energy performance as compared to the existing *reactive* fault-tolerant techniques. A scheduling technique is also proposed based on self-timed execution to minimize schedule construction and storage overhead. Experimental results indicate that the proposed approach achieves 95% less time at run-time for schedule construction. This is crucial in meeting real-time deadlines. Finally, the scheduling technique also minimizes the storage overhead by 92%, which is an important consideration, especially for multimedia applications.

CHAPTER 6

Run-time Adaptations for Lifetime Improvement

6.1 Introduction

As established in previous chapters, energy consumption and reliability are two of the most important optimization objectives in modern multiprocessor systems. There is a strong interplay between these two objective functions. Reducing the temperature of a system by efficient thermal management leads to a reduction of leakage power. On the other hand, a reduction of power dissipation (by controlling the voltage and frequency of operation) leads to an improvement in the thermal profile of a system. However, too frequent voltage and frequency scaling can lead to thermal cycles causing stress related reliability concern. This has attracted a significant attention in recent years to investigate on intelligent techniques, such as the use of machine learning, to determine the relationship between temperature, energy and performance and their control using voltage and frequency switching. The existing learning-based approaches suffer from the following limitations.

First, modern multicore systems switch between applications exhibiting wide performance and workload variations, and therefore the thermal behavior of these systems vary both within (intra) and across (inter) applications. Although intra-application thermal variations are considered in existing studies, inter-application variations are not ad-

dressed. Second, the existing studies focus on average and peak temperature reduction; thermal cycling is not accounted. Last, the existing adaptive techniques are either implemented on a simulator or rely on time-consuming thermal prediction from the *HotSpot* tool [68], limiting their accuracy and scalability.

In this work, the above gaps are addressed, and a dynamic thermal management approach is presented for multicore systems that adapts to thermal variations within (intra) and across (inter) applications. Fundamental to this approach is a run-time system, which interfaces with the on-board thermal sensors and uses reinforcement learning algorithm to learn the relationship between the mapping of threads to cores, the voltage-frequency of a core, and its temperature. The aim is to control the peak temperature, average temperature, and the thermal cycling to achieve an extended mean time to failure (MTTF). This work makes the following contributions:

1. inter-and intra-application thermal management using thread-to-core allocation (through CPU affinity[1]) and dynamic frequency control (through CPU governors[2]);

2. separation of the temperature sampling interval from the decision interval of the conventional reinforcement learning algorithm to accurately model (and hence control) the average temperature and thermal cycling; and

3. implementation of the run-time system incorporating the machine learning algorithm on a real platform.

The proposed approach is implemented on an Intel quad-core platform running Linux kernel 3.8.0. A set of multimedia applications from the *ALPBench* suite [154] are executed on the platform. Results demonstrate that the proposed approach minimizes average temperature and thermal cycling, leading to a significant improvement in MTTF as demonstrated in Section 6.5. Additionally, the static and the dynamic energy consumption are reduced by 11% and 10%, respectively.

The remainder of this chapter is organized as follows. Motivation of the thermal management for multicore systems is provided in Section 6.2. This is followed by an overview of the proposed reinforcement learning-based thermal management approach in Section 6.3 and the implementation details in Section 6.4. Evaluation of the proposed technique is presented next in Section 6.5, and the chapter is concluded in Section 6.6.

[1]CPU affinity enables the binding of a thread of an application to a physical core or a range of cores.
[2]CPU governors are power schemes for the CPU, deciding the frequency of operation of the cores.

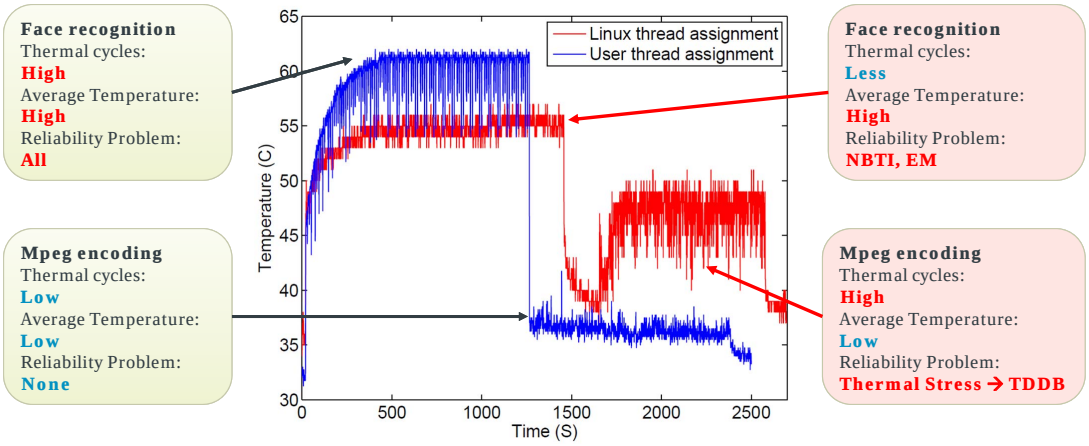


Figure 6.1: Thread-to-core affinity influences thermal profile.

6.2 Motivational Example

Thermal management using voltage and frequency control is already demonstrated in prior works (e.g. [135]). To establish the importance of thread allocation on the thermal behavior of applications, an experiment is conducted on an Intel quad-core platform by executing two multi-threaded (6 threads) applications (*face recognition* and *mpeg2 encoding*) back-to-back. The thermal profiles obtained using Linux's default thread-to-core allocation and scheduling is shown in red in Figure 6.1. From the thermal profiles it can be seen that *face recognition* is characterized by a higher average temperature with lower thermal cycling leading to peak temperature related reliability issues, such as electro-migration (EM) and negative bias temperature instability (NBTI). Application *mpeg2 encoding* on the other hand exhibits lower average temperature with higher thermal cycling leading to thermal fatigue and its associated reliability problems.

This difference in thermal behavior of the two different applications can be explained as follows. The thread workloads of the *face recognition* application are characterized by longer duration of thread-independent high activity cycles followed by shorter duration of inter-thread dependent low activity cycles. When these threads are allocated to cores, the longer independent high-activity cycles of a thread overlap partially with the shorter dependent low-activity cycles of other threads. This is due to the Linux's default thread allocation, where threads are often migrated to balance load on the architecture. This leads to a higher temperature with lower thermal cycling. For the *mpeg2 encoding* application, the thread-independent high-activity cycles are shorter in duration, while the inter-thread dependent cycles are relatively longer compared to the *face recognition* ap-

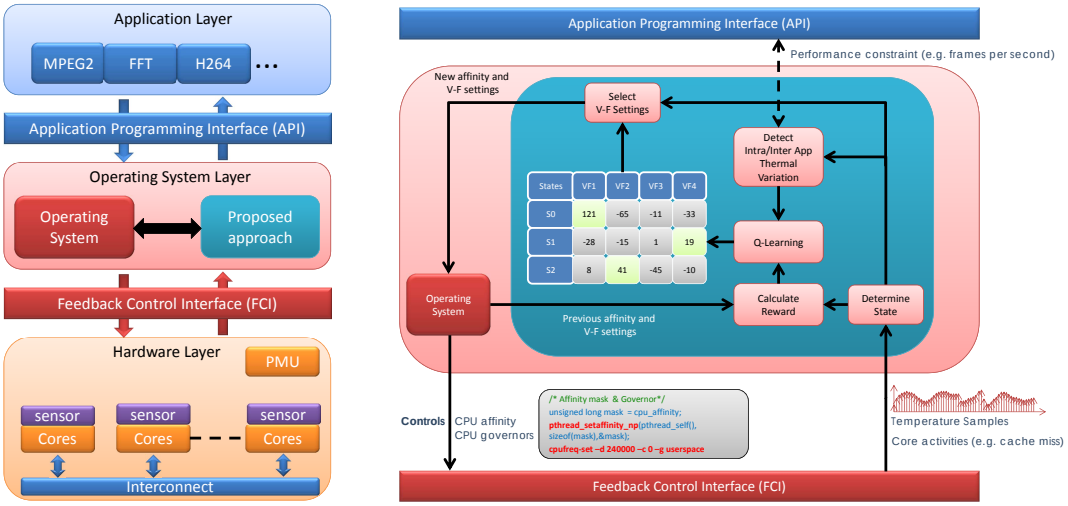
plication. When these threads are allocated by Linux, only few of the available cores are being used. The default allocation results in a combination of independent high-activity cycles (of more than one threads), which overlap with each other (leading to a higher temperature) and similarly, the longer low-activity inter-thread dependent cycles overlap (leading to a lower temperature). This results in alternating high and low temperatures triggering high thermal cycling.

Next, the same experiment is repeated by arbitrarily fixing the assignment of threads to cores (two cores execute two threads each and the other two cores execute one thread each) and leaving only the thread scheduling decision to the operating system. This is performed by changing all thread's affinity masks, forcing the Linux kernel to migrate these threads to the cores specified. The new thermal profiles are shown in blue in the same figure. As can be seen from the plot, this arbitrary thread assignment results in higher average temperature with higher thermal cycling for the *face recognition* application triggering all temperature related reliability concerns. This is because when threads are fixed to cores, the longer high-activity cycles of different threads overlap and so do the shorter low-activity cycles, resulting in higher temperature and higher thermal cycling. When the same control is applied for *mpeg2 encoding*, the shorter high-activity cycles are not combined but are overlapped with each other. The temperature increases, but for a shorter duration. This results in reduction of both the average temperature and thermal cycling thereby improving the lifetime. This example demonstrates two key aspects – thermal profile varies with application; and thread allocation influences thermal profile. This motivates the importance of an adaptive algorithm to learn the thermal behavior of an application and control it using appropriate thread-to-core assignment.

6.3 Proposed Thermal Management Approach

Figure 6.2a visualizes the three design layers typical of a modern multicore system. A general overview is provided on the interaction of these three layers.

Application Layer: The application layer is composed of a set of applications, which are executed on the system. Every application is characterized by a performance requirement, which determines the quality-of-service for the corresponding application. Table 6.1 reports the performance metric for some of the common applications for mod-



(a) Three design layers

(b) Q-learning based thermal optimization

Figure 6.2: Proposed thermal management approach.

Table 6.1: Performance metric of the common multicore applications.

Applications	Performance metric
MPEG2, MPEG4, H264	time to encode/decode a video frame
JPEG Enc/Dec	time to encode/decode an image
FFT/iFFT	time for 256 Fourier transforms
AES, SHA	time to hash 2048-byte message
basicmath, gzip, bitcount	time for 100 operations

ern multicore systems. An application is annotated to include its timing requirement, which is communicated to the operating system through the application programming interface. Additionally, the source code of the application is modified to insert breakpoints in order to signal the operating system to recalculate the thread affinities and the voltage-frequency values for the next execution interval. For video/image applications, breakpoints are inserted after encoding or decoding of every frame. Similarly for FFT/iFFT, the breakpoint interval is every 256 Fourier transforms. Figure 6.3 shows an example breakpoint-based program execution. The timing overhead τ_o incorporates the following: the time for the Q-learning algorithm to generate the new thread affinities and voltage-frequency values; time to migrate the application threads; and the time to set the voltage and frequency on the CPU cores through the operating system.

Operating System Layer: The operating system layer is responsible for coordinating the application execution on the hardware. Of the different responsibilities of the operating system, such as scheduling, memory management, and device management,

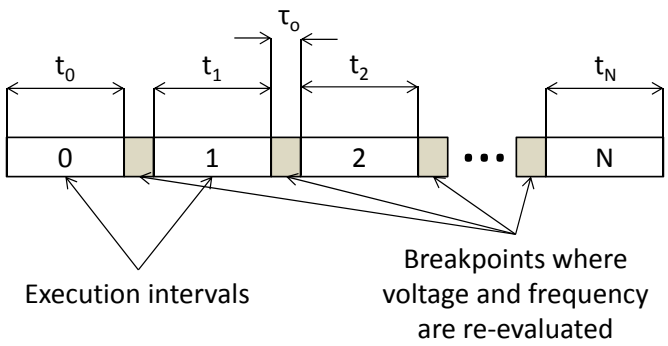


Figure 6.3: Breakpoints-based task execution.

the focus here is on the cross-layer interaction, which forms the background of the proposed machine learning-based thermal management. At every application breakpoint, the operating system stalls the application execution and triggers the proposed machine learning algorithm, which generates the new thread affinities and voltage-frequency values for the CPU cores. The operating system applies the new voltage-frequency values on the CPU cores using the `cpufreq-set` utility. The application threads are migrated to the corresponding cores, as specified using the affinity masks, using the operating system command `pthread_setaffinity_np`. The application execution interval is used to monitor the performance and the temperature. This interval is also referred to as *decision epoch* in the machine learning terminology. Throughout the rest of this work, the execution interval and the *decision epoch* are used interchangeably.

In most existing works on Q-learning based thermal management, the decision epoch is used to sample the temperature; actions are selected based on the instantaneous temperature from the sensor, which is not a true indication of the average temperature or thermal cycling in the interval. Since temperature-related reliability is governed by average temperature and thermal cycling, which need to be measured over a period of time, the temperature sampling interval is lower than than of the decision epoch for this work.

Hardware Layer: The hardware layer consists of the processing cores with thermal sensors and performance monitoring unit (PMU) to record the performance statistics. Of the different performance statistics available, this work considers *CPU cycles*, which gives a fair indication of the workload of a given application. The temperature samples are collected continuously by the operating system at the temperature sampling interval. The PMU readings are collected at every breakpoint and subsequently, the readings are reset to allow performance recording for the next execution interval. Finally, before the start of the next execution interval, the following sequence of events takes place:

- the application threads are migrated to the corresponding cores as specified using the `pthread_setaffinity_np` command;
- the frequency value, set by the operating system using the `cpufreq-set` command, is converted to a corresponding CPU clock divider setting;
- the divider settings are written into appropriate CPU registers; and
- the divided CPU clock is used to execute the migrated thread for the next execution interval.

6.4 Q-learning Implementation

The Q-learning [155] based thermal management approach is integrated in the operating system layer as discussed before. To give more insight into the approach, Figure 6.2b shows its internal details. The different components of the proposed approach are discussed next starting with a general overview of the Q-learning algorithm.

6.4.1 General Overview

The Q-learning is a reinforcement learning technique used to find the optimum policy of a given Markov Decision Process (MDP). A simplistic view of the Q-learning is shown in Figure 6.4. The algorithm consists of a learning agent that works by observing the state (s) of the environment and selecting a suitable action (a) to control the state. The learning is quantified and stored in a table (referred to as Q-Table in machine learning terminology) corresponding to every state-action pair. Every entry of this table, corresponding to the state-action pair (s, a), represents the reward (or penalty, if the entry is negative) obtained by selecting action a when the environment is in state s. The algorithm works by always selecting the action with the highest reward for any observed state of the environment.

In the context of multicore thermal management, the learning philosophy is as follows:

S1: determine the previous state-action pair;

S2: update the Q-Table entry for this state-action pair based on the current state;

S3: select the thread affinities and voltage-frequency values based on the current state;

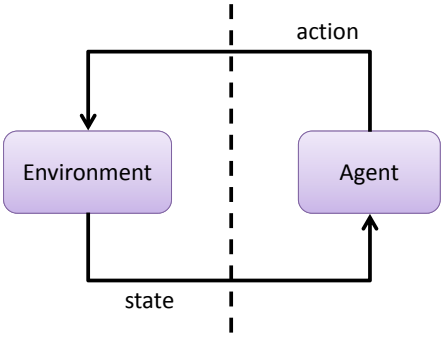


Figure 6.4: Basic Q-Learning.

S4: apply the selected action for the next execution interval.

Referring back to Figure 6.2b, step **S1** is implemented in the *Calculate Reward* block, step **S2** in the *Q-learning* block, step **S3** is implemented in the *Select V-F Settings* block and finally step **S4** in the *Operating System* block. The block *Detect Intra/Inter App Thermal Variation* is used to update the Q-Table.

6.4.2 Determine State

As established in Section 6.3, the Q-learning based thermal management is triggered at every application breakpoint. The *Determine State* block of the approach determines the current state of the system from the temperature readings of the thermal sensors. These temperature readings are used to calculate the thermal *aging* $\mathcal{A}$ (refer to Equation 2.21) and thermal *stress* $\mathcal{S}$ (refer to Equation 2.11). To limit state space explosion, the working range of these parameters are divided into N_a and N_s disjoint intervals, respectively. Specifically, *stress* is the set $\mathcal{S} = \{(0, s_0], (s_0, s_1], \cdots, (s_{N_s-1}, s_{N_s}]\}$ and the symbol $\hat{\mathfrak{s}}_i$ is used to represent the interval $(s_i, s_{i+1}]$. Similarly, *aging* is the set $\mathcal{A} = \{(0, a_0], (a_0, a_1], \cdots, (a_{N_a-1}, a_{N_a}]\}$ and the symbol $\hat{\mathfrak{a}}_i$ is used to represent the interval $(a_i, a_{i+1}]$. The environment is represented as $\mathcal{E} : (\mathcal{A} \times \mathcal{S})$.

6.4.3 Action Space of the Q-learning Algorithm

As shown in Section 6.2, the thermal behavior of a multicore system is influenced by CPU affinity and the voltage-frequency settings. These forms the action space of the Q-learning algorithm, which is represented by $A : (\mathcal{M} \times \mathcal{V})$ where $\mathcal{M}$ is the set of thread affinity mappings and $\mathcal{V}$ is the set of governors.

6.4.4 Q-learning Algorithm

The Q-learning block in Figure 6.2b is responsible for updating the Q-Table entries. The reward or penalty for selecting an action at a state is calculated at the next breakpoint based on the goodness of the selected action. In other words, at each breakpoint, the state-action pair of the last execution interval is evaluated. For this, the last selected action is read in a local variable `laction`, through the operating system commands `cpufreq-info` and `pthread_getaffinity_np`. The last state is stored in another local variable `lstate`. This variable is updated with the current state, once its old value is used to update the Q-Table entry. The goodness of the last action in the last state is measured as reward (or penalty if it is negative) and is calculated based on

- *the performance slack* i.e. the difference between the time required to execute the last execution interval and the timing requirement specified in the annotated application source code; and
- *the thermal safety* i.e. the current thermal state of the system.

Mathematically, this is represented as

$$r = \begin{cases} -\hat{\mathfrak{s}}_i \times \hat{\mathfrak{a}}_i & \text{if } (\hat{\mathfrak{s}}_i = \hat{\mathfrak{s}}_{N_s}) \text{ or } (\hat{\mathfrak{a}}_i = \hat{\mathfrak{a}}_{N_a}) \\ f(\hat{\mathfrak{a}}_i, \hat{\mathfrak{s}}_i) + (P_c - P) & \text{otherwise} \end{cases} \tag{6.1}$$

where P is the performance, P_c is the performance constraint, and the function f is determined empirically as $f = (a.K_1.stress + b.K_2.aging)$, where a and b are relative importance of *stress* and *aging*: For mpeg (large thermal cycles), $a > b$ and for tachyon (high average temperature), $b > a$. Two sets of a and b values are used based on the mean of *stress* and *aging*. $K_1(K_2)$ is the learning weight and is a Gaussian function of the *stress* (*aging*) values. This distribution assigns lower rewards to thermally unstable as well as the thermal stable states and thus, allows the algorithm to explore other states and prevent Q-Table clustering.

For the design of the reward function, two cases are considered. If the *stress* or *aging* falls in the unsafe zone (the last interval), the decision is penalized. This is indicated with a negative value of the reward function. For all other cases, the reward function is composed of performance penalty and the thermal safety of the state. Specifically, if the performance requirement is not satisfied, $(P_c - P)$ is negative and the reward (or

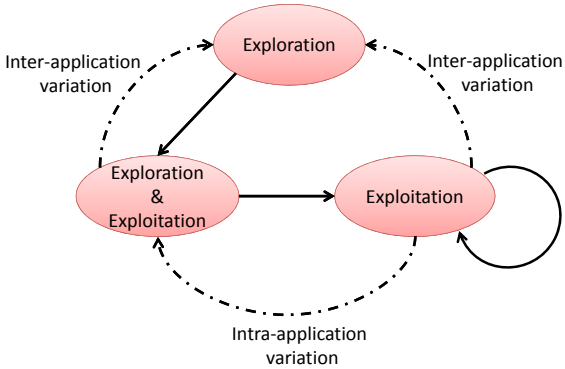


Figure 6.5: Three phases of the Q-learning algorithm.

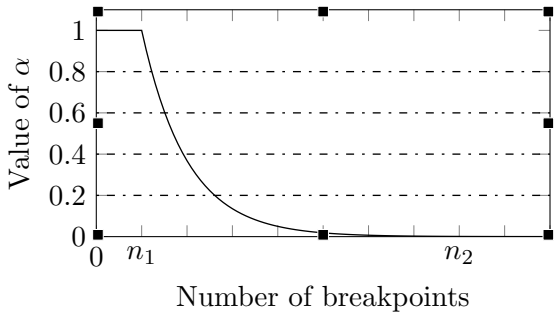


Figure 6.6: Change of α with number of breakpoints.

penalty) is governed by the function f. Finally, rewards are guaranteed if an action leads to a thermal safe state while satisfying the performance requirements.

The reward value of Equation 6.1 is used to update the Q-table entry using the following equation.

$$Q(\texttt{lstate}, \texttt{laction}) = Q(\texttt{lstate}, \texttt{laction}) + \alpha \cdot r_i \tag{6.2}$$

where α is the learning rate. One of the interesting feature of the Q-learning algorithm is its three phases of operation. This is shown in Figure 6.5. These three phases are *exploration*, *exploration-exploitation*, and *exploitation*. In the *exploration phase*, the algorithm learns the goodness of all the different actions. Therefore, the table entries are updated with all of the calculated reward values (Equation 6.1). This phase is characterized by $\alpha = 1$. On the other hand, in the *exploitation* phase, the algorithm selects the action based on the previous learning. Therefore, the table entries are not updated with the reward value. This phase is characterized by $\alpha = 0$. In between these two phases, the algorithm remains in the *exploration-exploitation* phase. In this phase, the algorithm learns the goodness of only the good actions (outcome of the *exploration* phase) to evaluate it further. This phase is therefore characterized by $0 < \alpha < 1$.

Figure 6.6 plots the change in the alpha values for these three phases against the number of invocations of the Q-learning algorithm i.e. the number of breakpoints. The *exploration* phase is active for 0 to n_1 breakpoints, the *exploration-exploitation* phase is active for n_1 to n_2 breakpoints and the *exploitation* phase is active beyond n_2 breakpoints. Let N denote the total number of breakpoints to be inserted in the application execution (refer to Figure 6.3). The following lemma can be stated.

Lemma 1 *The convergence of any Q-learning algorithm is guaranteed if* $N \geq n_2$.

In the context of this work, the above lemma states that for thermal management using Q-learning algorithm, the number of breakpoints to be inserted in the application execution should be such that the algorithm reaches the *exploitation* phase. However, too many breakpoints during a small application execution will incur performance penalty. The performance parameters in Table 6.1 are defined based on these considerations.

The state transition diagram of the unmodified Q-learning is shown in Figure 6.5 with the dark solid lines. To introduce autonomous reaction of the algorithm to intra- and inter-application thermal variations, the state transition diagram is modified using the dotted lines as shown in the figure. When an intra-application thermal variation is detected, the modified algorithm transits to the *exploration-exploitation* phase and the α value is changed accordingly. On the other hand, for inter-application variation, the modified algorithm transits to the *exploration* phase resetting $\alpha = 1$. The detection of the intra- and inter-application thermal variations is discussed next.

6.4.5 Detection of Intra- and Inter-Application Thermal Variations

To incorporate intra- and inter-application thermal variations, moving averages of the *stress* and the *aging* are determined at every breakpoint. The change in the moving averages are identified as ΔMA_s and ΔMA_a. Two thresholds are maintained for each of these quantities identified with the superscript L and U, respectively. A change in the moving average is considered as intra-application variation, if the change is greater than the lower threshold and lower than the upper threshold (for example when $\Delta MA_s^L \leq \Delta MA_s < \Delta MA_s^U$). On the other hand, the change in the moving average is considered as inter-application variation, if the change is greater than the upper threshold.

Table 6.2: Temperature (°C) of the proposed approach.

Benchmarks		Average Temperature (°C)			Peak Temperature (°C)		
Application	**Data**	**Linux [158]**	**Ge et. al [135]**	**Proposed**	**Linux [158]**	**Ge et. al [135]**	**Proposed**
tachyon	set 1	69.2	52.6	50.6	71.5	63.0	60.0
	set 2	50.5	44.5	43.8	57.3	56.3	52.0
	set 3	50.8	44.7	41.6	57.8	54.5	48.8
mpeg dec	clip 1	36.0	34.0	34.2	42.7	41.3	39.0
	clip 2	35.6	34.4	34.2	42.3	42.0	39.3
	clip 3	34.3	34.4	34.0	43.0	39.7	44.3
mpeg enc	seq 1	33.7	34.1	32.6	41.0	40.7	40.3
	seq 2	34.4	33.5	32.3	41.3	39.7	41.7
	seq 3	33.2	33.7	31.8	40.3	40.0	41.0

6.5 Results

The proposed run-time approach is validated experimentally on an Intel quad-core CPU running Linux kernel 3.8.0. Performance is monitored using *perf* [156]; temperature is measured by sampling the thermal sensors directly; power/energy consumption is recorded using *likwid powermeter* [157]. A set of multi-threaded multimedia applications are considered from the *ALPBench* [154] benchmark suite. These benchmarks are *mpeg enc, mpeg dec, face recognition, sphinx*, and *tachyon*. These are representative of the multimedia workloads for most multicore systems. The number of threads in each of these applications is set to six. The device parameters used for computing the *aging* and *stress* of a core are the same as that used in [97, 126].

6.5.1 Intra-Application

Table 6.2 reports the average temperature and peak temperature and Table 6.3 reports the MTTF due to average temperature (equivalently *aging*) and thermal cycling (equivalently *stress*) of the proposed technique in comparison with Linux's *ondemand* [158] and the technique proposed in [135]. Results are reported for three different applications, each of which is executed for three sets of input data. Furthermore, the minimum of the two MTTF values are also reported in Table 6.3 (columns 9 - 11).

There are a few trends to follow from this table. First, the technique in [135] minimizes instantaneous temperature achieving a lower *aging* (higher MTTF) than Linux (refer to Table 6.2 columns 3-4 & Table 6.3 columns 6-7). This signifies the importance of the thermal management feature for operating systems. However, thermal cycling is not accounted for in this technique and therefore does not guarantee reduction of *stress*. This is evident from the thermal cycling-related MTTF values (Table 6.3 columns 3-4) for scenarios such as *tachyon* on set 1 and *mpeg dec* on clip 1, where the MTTF values

Table 6.3: MTTF (in years) of reinforcement learning algorithm for three applications. The scaling parameters for computing MTTF are so selected such that the MTTF of an unstressed core (i.e. an idle core) is 10 years.

Benchmarks		**Thermal cycling MTTF**			**Thermal aging MTTF**			**Overall MTTF**		
Application	**Data**	**Linux [158]**	**Ge et al. [135]**	**Proposed**	**Linux [158]**	**Ge et al. [135]**	**Proposed**	**Linux [158]**	**Ge et al. [135]**	**Proposed**
tachyon	set 1	7.1	2.3	5.5	0.7	3.0	3.6	0.7	2.3	3.6
	set 2	2.8	4.3	5.3	2.6	4.5	4.8	2.6	4.3	4.8
	set 3	1.3	3.8	6.5	2.4	4.1	5.5	1.3	3.8	5.5
mpeg dec	clip 1	2.1	0.8	6.4	3.7	4.5	4.4	2.1	0.8	4.4
	clip 2	1.1	0.9	4.7	3.8	4.3	4.4	1.1	0.9	4.4
	clip 3	1.6	3.4	3.7	4.3	4.2	4.5	1.6	3.4	3.7
mpeg enc	seq 1	4.3	4.4	5.2	4.6	4.5	5.2	4.3	4.4	5.2
	seq 2	3.9	6.2	4.8	4.3	4.7	5.4	3.9	4.7	4.8
	seq 3	4.6	5.1	5.1	4.9	4.6	5.7	4.6	4.6	5.1

obtained using [135] are lower than that of Linux.

Second, the proposed reinforcement learning algorithm minimizes the average temperature by upto 18.6^oC and the peak temperature by upto 11.5^oC. This reduction leads to an improvement in MTTF due to *aging* by upto 5x (average 82%) as reported in Table 6.3 column 8. For applications such as *mpeg dec*, the improvement is less as the average temperature is usually lower, leading to a limited scope to further improve *aging*. The thermal cycling effect dominates in this application. For other applications such as *tachyon*, the improvement is significant due to the large scope for improving both *aging* and *stress*. Third, the proposed adaptive algorithm also minimizes thermal cycling which is not considered in Linux and [135]. This is highlighted in Table 6.3 column 5. An important point to note is that, for the *tachyon* application with set 1 data, the MTTF due to *stress* using Linux's default thread assignment is higher (7.1 years); however, the MTTF due to *aging* is lower (0.7 years). The proposed technique balances the two effects and improves the MTTF due to *aging* by 5x with less than 25% sacrifice in MTTF due to *stress* (while still maintaining a satisfactory MTTF of 5.5 years). For all other applications and data sets, the proposed reinforcement learning algorithm outperforms Linux in terms of thermal cycling by an average 2.3x.

Last, the proposed approach outperforms [135] both in terms of *aging* (an average 13% higher average temperature related MTTF) and *stress* (an average 2x higher thermal cycling related MTTF). While the improvement of thermal cycling is expected (as this is incorporated explicitly in the proposed approach), the improvement in *aging* is due to the following. First, the decoupling of the temperature sampling interval from the decision epoch (enabling a finer control on the average temperature); second, careful choice of the

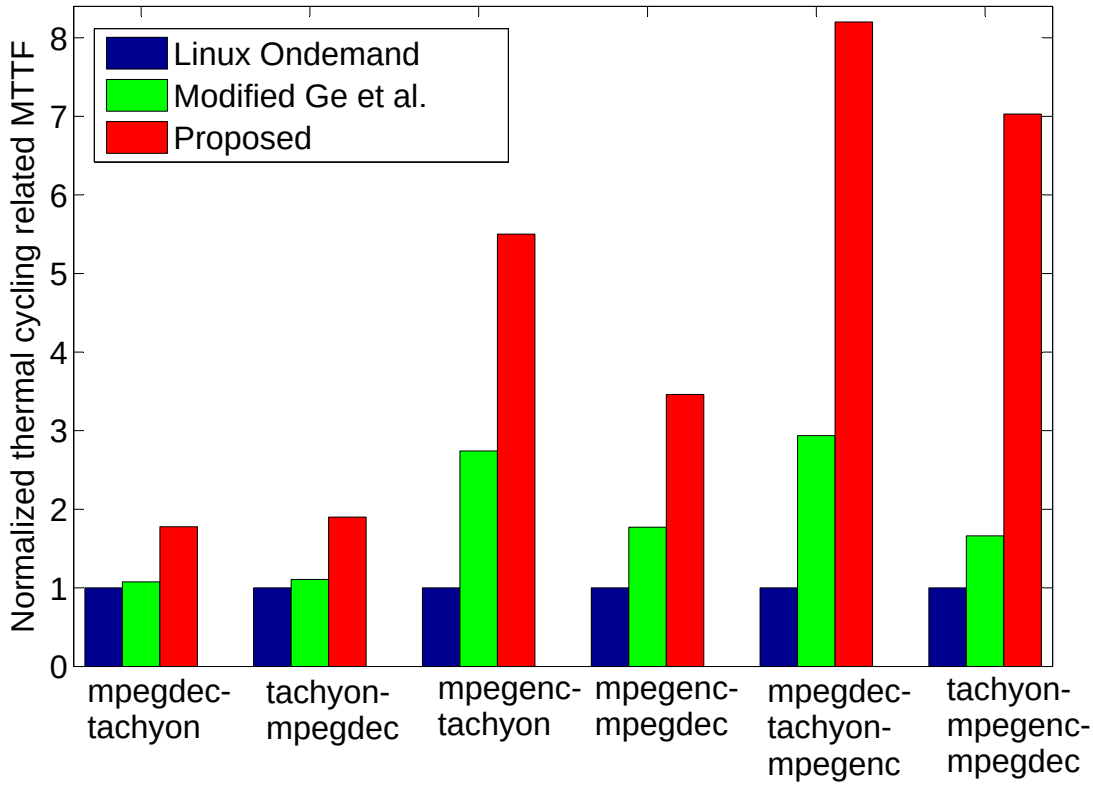


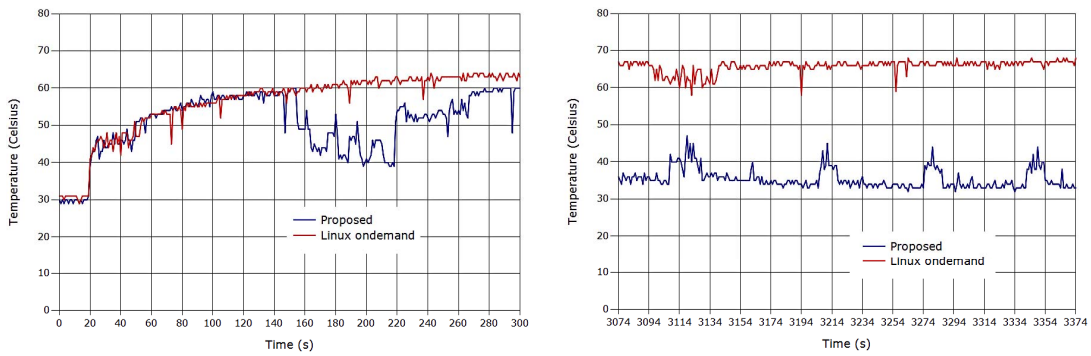
Figure 6.7: Inter-application results.

design parameters as demonstrated in Figure 6.9 and 6.10.

6.5.2 Inter-Application

Figure 6.7 plots the normalized MTTF due to thermal cycling obtained using the proposed technique in comparison with that obtained using the modified technique of [135] (referred in this figure as *Modified Ge et al.*) for six different inter-application scenarios. The MTTF values are normalized with respect to the MTTF obtained using Linux's *on-demand* governor. Furthermore, the technique of [135] is modified to consider application switching using explicit indication from the application layer. The proposed approach however, detects application switching autonomously (without communication from the application layer) and performs re-learning (as discussed in Section 6.4).

There are six inter-application scenarios considered in this experiment. A scenario *appA-appB* indicate that *appA* is executed first followed by application *appB*. As can be seen from the figure, the technique of *Ge et. al* [135] results in higher MTTF than Linux. For some inter-application scenarios such as *mpegdec-tachyon* and *tachyon-mpegdec*, the improvements are less ($\approx 8\%$). For other inter-application scenarios, the improvement is higher. On average for all the scenarios, [135] increases MTTF by 80% as compared to



(a) Exploration phase of the learning algorithm. (b) Exploitation phase of the learning algorithm.

Figure 6.8: Phases of the Q-learning algorithm.

Linux. The approach proposed in this work outperforms both Linux and that of [135] in terms of thermal cycling, achieving 5x improvement with respect to Linux and 3x improvement with respect to [135]. A point to note from the figure is that, the MTTF improvement (over [135]) using the proposed approach, increases with frequent application switching. This is evident from the higher MTTF improvement of 3.5x obtained for the three-application scenarios (*mpegdec-tachyon-mpegenc* and *tachyon-mpegenc-mpegdec*) as compared to improvements obtained for the four other two-application scenarios. To conclude, the proposed approach improves thermal cycling related MTTF significantly for inter-application scenarios. The improvement increases with an increase in application switching (typical of a modern multicore system).

6.5.3 Phases of the Reinforcement Learning Algorithm

To further demonstrate the temperature profile obtained using the proposed algorithm, Figures 6.8a and 6.8b plot the exploration and the exploitation phases, respectively in comparison with Linux's popular and default *ondemand* governor for the *face recognition* application. As can be seen, at the beginning of the exploration phase, the temperature obtained using the proposed algorithm is comparable to that obtained using Linux. This is because, at this phase, the Q-learning algorithm explores the impact of CPU affinity and operating frequency choices on the temperature of a core. However, when the algorithm learns the impact of these parameters on temperature (i.e. in the exploitation phase), the CPU affinities and operating frequencies are selected such that the average temperature is reduced (Figure 6.8b).

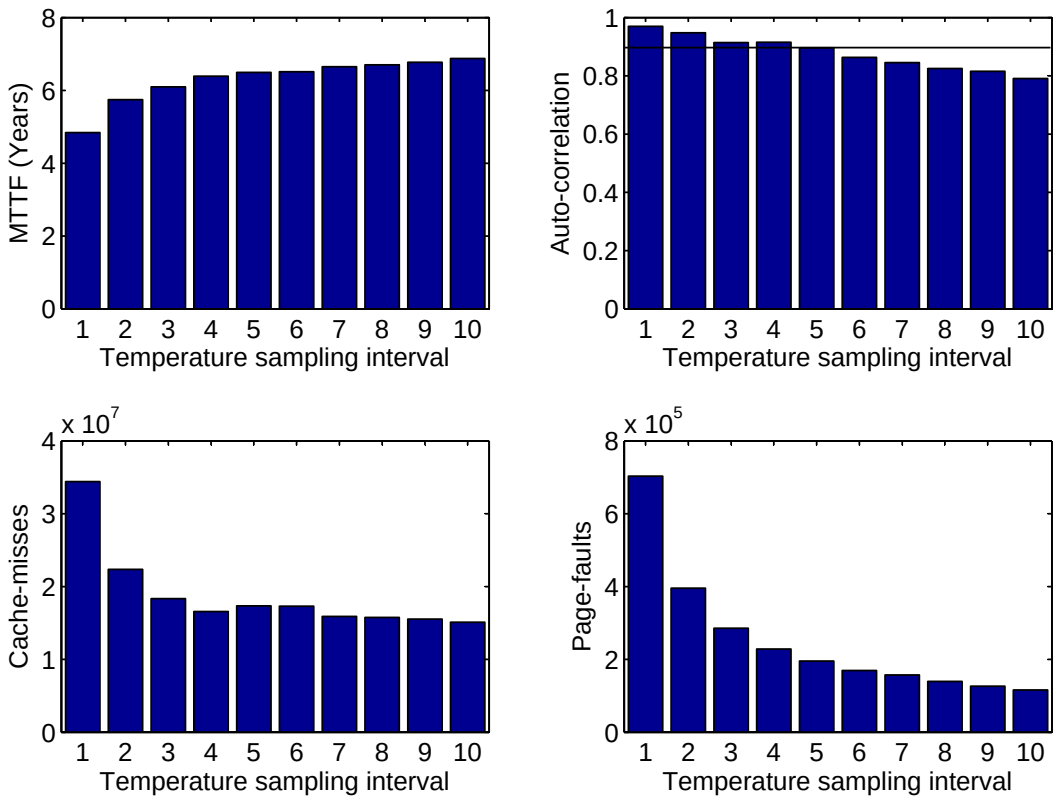


Figure 6.9: Impact of temperature sampling interval (sec).

6.5.4 Selection of Design Parameters

Figure 6.9 plots the impact of varying the temperature sampling interval on the *thermal stress* and the performance, for the *tachyon* application. The figure also plots the auto-correlation of temperature samples, which is a measure of how much the temperature values change across consecutive samples. Specifically, a high auto-correlation indicates a small difference between the consecutive thermal data. As can be seen from the figure, the auto-correlation is high for low sampling interval. This is expected because temperature variation is usually slower, being dependent on the thermal property of silicon, the ambient temperature, and the cooling technology used. The MTTF value increases with an increase in size of the sampling interval. This is, however, an over estimation of the actual MTTF value (corresponding to the sampling interval of 1 sec). This over-estimation is due to the loss of temperature accuracy with increasing sampling interval, resulting in a low *stress* and hence high MTTF. Finally, the number of cache-misses and page faults decrease with an increase in the sampling interval, clearly signifying the performance improvement. Based on these trade-offs, a sampling interval of 3 sec is selected for the *tachyon* application. Similarly, the sampling interval for other applications are determined. An interval of 3 sec provides the best trade-off for most applications. In

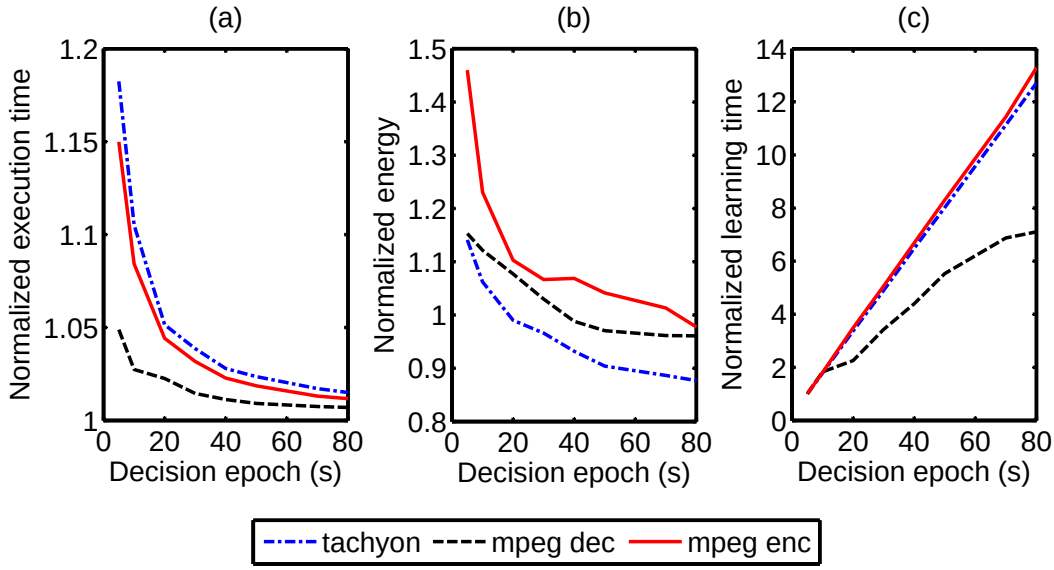


Figure 6.10: Effect of the length of decision epoch.

future, determination of the sampling interval can be incorporated as part of the learning algorithm itself.

Figure 6.10 plots the performance of the algorithm with increasing decision epochs for three applications. The execution time, dynamic energy consumption and the adaptation time are compared. The execution time refers to the completion time with a fixed length of input data. For the *mpeg enc* (or *mpeg dec*) application, this is the time to encode(decode) 10MB of video. For the *tachyon* application, the time is measured as the rendering time of 300 images. The execution time using the reinforcement learning algorithm is normalized with respect to the execution time of Linux (with no adaptation) for the same input data. As can be seen from Figure 6.10a, for all these applications, the execution time overhead is higher for smaller decision epochs due to the overhead associated with frequent decision changes. The execution time overhead reduces with larger decision epochs. Figure 6.10b reports the dynamic energy consumption of the proposed approach for the same input data normalized with respect to the Linux (without adaptation). As expected, the energy consumption is also higher for smaller decision epochs due to the frequent adaptation of the approach. Finally, Figure 6.10c plots the training time i.e. the time required for the algorithm to learn the thermal behavior of an application. The results are normalized with respect to the training time of the algorithm with a decision epoch of 5sec. As can be seen, the training time increases with an increase in the decision epoch. This is because, the training time is a function of decision epoch and

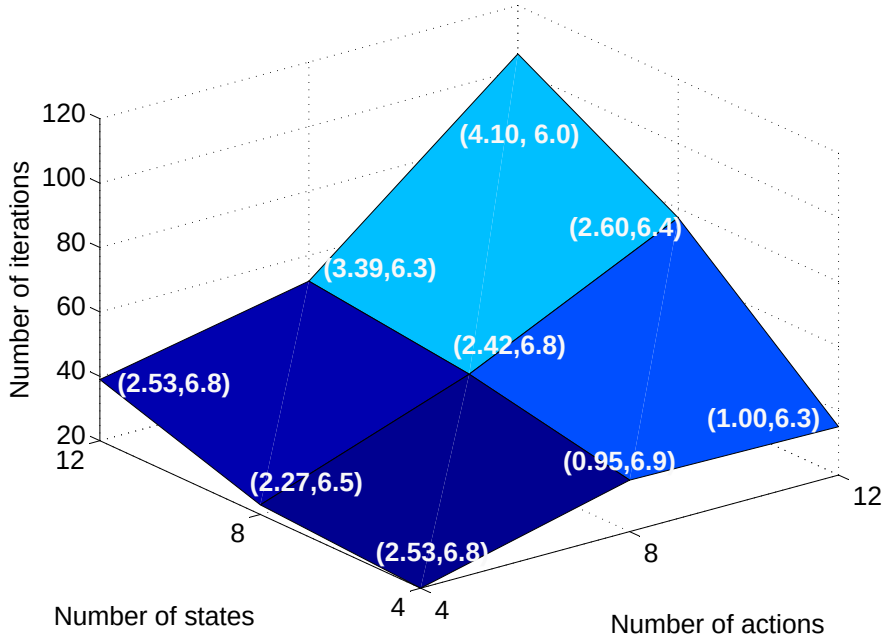


Figure 6.11: Convergence of the reinforcement learning algorithm.

number of iterations N. Therefore, for a constant N, the training time increases with the decision epoch. The decision epoch of the proposed algorithm is selected based on this energy, execution time, and training time trade-offs.

Figure 6.11 plots the convergence time of the proposed algorithm for the *mpeg decoding* application with a varying number of states and actions. The convergence time is measured by the number of decision epochs needed to train the proposed learning algorithm. As can be seen from the figure, the number iterations i.e., the training time increases with an increase in the number of actions and states. This is expected because, an increase in the number of states or actions leads to an increase in the size of the Q-Table and therefore more iterations are needed to learn (fill the table entries). The figure also reports the MTTF as coordinates (*stress*, *aging*) for each design point. As the size of the Q-Table increases, the reinforcement learning algorithm has finer control on the temperature, resulting in an improvement in the MTTF. The number of states and actions are chosen based on this learning time and solution quality trade-off.

6.5.5 Performance and Energy Trade-offs

Table 6.4 reports the execution time of the proposed approach in comparison with the one proposed in [135] and the Linux-based approach for *ondemand*, *powersave* and *user-space* power governors. Two user frequencies (2.4GHz and 3.4GHz) are shown in the

Table 6.4: Execution time (in sec) of the proposed approach.

Application	Linux				Ge et. al [135]	Proposed
	ondemand	powersave	2.4GHz	3.4GHz		
tachyon	629	1337	913	627	1137	810
mpeg dec	1208	1222	1183	1127	1328	1204
mpeg enc	1623	1655	1628	1571	1676	1599

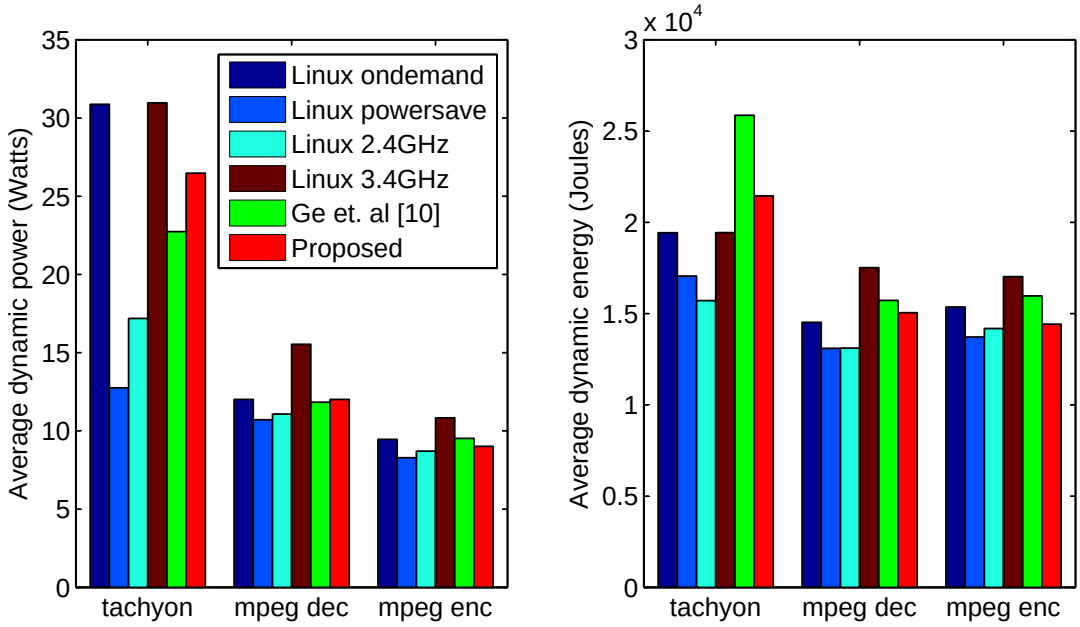


Figure 6.12: Power comparison of the reinforcement learning algorithm.

table. The execution time with the highest frequency of 3.4GHz is the least for all the applications. This is expected because the execution time decreases with an increase in the frequency. For the same reason, the execution time with the lowest frequency (*powersave*) is the highest. However, the power overhead is the least (as explained next). For some applications such as *tachyon*, the proposed approach has higher execution time than the Linux's *ondemand* governor by upto 30%. This is because the workload in the *tachyon* application forces the Kernel to execute always at the highest frequency of 3.4GHz in the *ondemand* mode. Thus, the execution time for *ondemand* and *userspace-3.4GHz* are comparable. The proposed approach on the other hand explores different power modes to reduce thermal *stress* and *aging* and therefore trades-off performance. For other applications such as the *mpeg enc* and the *mpeg dec*, the execution time of the proposed approach is lower than that of the *ondemand* governor. Finally, with respect to [135], the proposed approach reduces execution time by an average 14%.

Figure 6.12 plots the average dynamic power and energy consumption (measured

using *likwid-powermeter*) of the proposed algorithm in comparison with that of [135] and the Linux governors. Although the power and energy overhead are not reported in [135], these are calculated here for a comparative study. As can be seen from the figure, the proposed approach reduces power consumption by an average 6% in comparison with Linux *ondemand* governor with 10% increase in execution time. Although the dynamic power consumption of [135] is lower than the proposed approach (on average 4% lower), the energy consumption (which incorporates both power and performance) of the proposed approach is 10% lower than [135] (within 3% of the energy consumption of Linux's *ondemand* governor). An interesting point to note here is that, by reducing the average temperature, the proposed technique improves the leakage power. Although leakage power is not measured on the board, the established models for the same (such as the one proposed in [97]) indicate that the proposed algorithm improves leakage energy by an average 15% as compared to the *ondemand* governor and 11% as compared to [135].

6.6 Remarks

In this work, a reinforcement learning-based run-time approach is proposed for multicore system to adapt to thermal variations both within an application as well as when the system switches from one application to another. The control is provided by overriding the operating system mapping decisions using affinity masks and dynamically changing the frequency of cores using CPU governors. The approach is validated experimentally using an Intel quad-core platform running Linux kernel 3.8.0. Results demonstrate that the proposed approach is able to improve MTTF by an average 2x for intra-application and 3x for inter-application scenarios as compared to the existing dynamic thermal management technique. Furthermore, the approach also improves dynamic energy consumption by an average 10% and static energy by 11%.

CHAPTER 7

Conclusions and Future Directions

Reliability and energy are emerging as two of the growing concerns for multiprocessor systems at deep sub-micron technology nodes. This thesis presented a system-level approach, namely application mapping and scheduling, to jointly address the reliability and energy problems for multiprocessor systems. A detailed background is presented in Chapter 2 on synchronous data flow graphs (SDFGs), used as application models in this thesis. This chapter also presented the mathematical background on lifetime reliability and the related studies on task mapping and scheduling for lifetime improvement.

In Chapter 3, a platform-based design methodology is proposed that involves task mapping on a given multiprocessor system to jointly minimize energy consumption and temperature related wear-out, while satisfying the performance requirement. Fundamental to this methodology is a simplified temperature model that incorporates not only the transient and steady-state behavior (temporal effect), but also its dependency on the temperature of the surrounding cores (spatial effect). The proposed temperature model is integrated in a gradient-based fast heuristic that controls the voltage and frequency of the cores to limit the average and peak temperature leading to a longer lifetime, simultaneously minimizing the energy consumption. A linear programming approach is then proposed to distribute the cores of a multiprocessor system among concurrent applications (use-cases) to maximize the lifetime. Experiments conducted with a set of synthetic

and real-life applications represented as SDFGs demonstrate that the proposed approach minimizes energy consumption by an average 24% with 47% increase in lifetime.

In Chapter 4 of this thesis, a fast design space exploration technique is presented for hardware-software partitioning by analyzing the negative impact of increasing the number of checkpoints for transient fault-tolerance on the lifetime reliability of the processing cores. Based on this, a hardware-software co-design approach is proposed to determine the architecture of a reconfigurable multiprocessor system to maximize its lifetime reliability by considering applications enabled individually and concurrently. Experiments conducted with real life and synthetic applications represented as SDFGs on a reconfigurable multiprocessor system demonstrate that the proposed technique improves lifetime reliability by an average 65% for single applications and an average 70% for use-cases with 25% fewer GPPs and 20% less reconfigurable area as compared to the existing hardware-software co-design approaches.

To ensure graceful performance degradation in the presence of faults, a design-time (offline) multi-criterion optimization technique is proposed in Chapter 5 for application mapping on embedded multiprocessor systems to minimize energy consumption for all processor fault-scenarios. A scheduling technique is then proposed based on self-timed execution to minimize the schedule storage and construction overhead at run-time. Experiments conducted with SDFGs on heterogeneous multiprocessor systems demonstrate that the proposed technique minimizes energy consumption by 22% and design space exploration time by 100x, while satisfying the throughput requirement for all processor fault-scenarios. For scalable throughput applications, the proposed technique achieves 30% better throughput per unit energy, compared to the existing techniques. Additionally, the self-timed execution-based scheduling technique minimizes schedule construction time by 95% and storage overhead by 92%.

Finally, an adaptive thermal management approach is proposed in Chapter 6 as a part of the run-time methodology, to improve the lifetime reliability of multiprocessor systems by considering both inter- and intra-application thermal variations. Fundamental to this approach is a reinforcement learning algorithm, which learns the relationship between the mapping of threads to cores, the frequency of a core and its temperature (sampled from on-board thermal sensors). Action is provided by overriding the operating system's mapping decisions using affinity masks and dynamically changing CPU frequency us-

ing in-kernel governors. Lifetime improvement is achieved by controlling not only the peak and average temperatures, but also thermal cycling, which is an emerging wear-out concern in modern systems. The proposed approach is validated experimentally using an Intel quad-core platform executing a diverse set of multimedia benchmarks. Results demonstrate that the proposed approach minimizes average temperature, peak temperature and thermal cycling, improving the mean time to failure by an average of 2x for intra-application and 3x for inter-application scenarios when compared to existing thermal management techniques. Additionally, the dynamic and static energy consumption are also reduced by an average 10% and 11%, respectively.

7.1 Near and Far Future Challenges

While this thesis presented the design methodologies for multiprocessor systems to jointly optimize reliability and energy consumption, a number of issues remain to be solved. Some of these are highlighted below.

7.1.1 Future Challenges for Design-time Analysis

Heterogeneity challenges: One of the key challenges associated with heterogeneous architectures involve task mapping. The execution times of a task on different processors are different. Further, not every task can be executed on every processor. This makes the task mapping and scheduling problem difficult to solve. The task migration approach also needs to be re-visited. This is because the new object code of a task needs to be compiled and transferred to the migrating core, if this is of different type than the core where the task was executing initially. Storing the pre-compiled object code of all tasks for all core type on every core incurs significant storage overhead and is crucial for multimedia multiprocessor systems where storage space is limited. Further, the transfer of object code on the networks-on-chip could lead to potential congestion of the same and could interfere with the data communication among the non-faulty cores. These challenges need to be solved for the existing design-time based approaches.

Dynamic partial reconfiguration: This thesis has shown that, there exists a significant scope of reliability and energy improvement for FPGA-based reconfigurable multiprocessor systems. Although reconfiguration of the FPGA logic is allowed before enabling

an application on the system, it is anticipated that further improvement is possible by allowing dynamic partial reconfiguration i.e., by allowing a part of the FPGA logic to be programmed for an application, while the remaining part of the logic is in operation. This will allow to partition more tasks as hardware tasks (i.e., tasks that will be executed on the FPGA logic) during the hardware-software partitioning stage. This will reduce the stress on the processing cores, increasing their lifetime reliability. However, dynamic partial reconfiguration involves timing overhead to program the FPGA bits. Thus, complex analysis frame-work needs to be developed to predict the performance and improve lifetime reliability. Additionally, dynamic voltage and frequency scaling capabilities is to be explored, which is not addressed currently for reconfigurable multiprocessor systems.

3D multiprocessor system: Most of the existing reliability optimization techniques are limited to 2D multiprocessor systems. Recent developments on 3D systems have attracted a significant research attention towards resource allocation and management for 3D systems. These systems offer significant performance and energy advantages over the native 2D multiprocessor systems, but is known to suffer from thermal issues. An effective approach to solve this challenge involves a thorough understanding of the reliability concerns for these systems, including the detailed transistor-level reliability modeling, which is still at an early stage of development; the thermal characterization of 3D systems; and finally the resource allocation for reliability improvement.

7.1.2 Future Challenges for Run-time Analysis

The proposed run-time thermal management for multiprocessor systems addressed intra- and inter-application workload variations. The next step towards this is to solve the following three key challenges – accuracy, scalability and scope.

Accuracy: Modern commercial off-the-self multiprocessor systems, such as Intel Ivy Bridge or Haswell architectures, are equipped with cooling mechanisms in the form of heat sink and fan. The temperature of the different cores of the architecture varies according to the fan speed and the heat sink mechanism. The temperature is also dependent on the placement of these cooling mechanisms in the floorplan. Specifically, cores located closer to the heat sink are often cooler than other cores, even when all the cores execute the same workload. Additionally, the speed of the fan can also be controlled by controlling the current drawn and therefore, the heat dissipation can be adapted based

on the application workload. An efficient thermal management needs to incorporate (1) the floorplan information and (2) workload-aware cooling control, alongside the existing techniques, such as dynamic voltage and frequency scaling (DVFS) and task mapping.

Scalability: The future of multiprocessor systems is many-core architecture, integrating hundreds of cores on the same die. The shared memory architecture for multi-core system is expected to become a bottleneck for many-core systems, and therefore message passing interface (MPI) is expected to become the standard communication protocol for many-core architectures. The existing techniques for multi-core systems need to be re-visited from the MPI perspective. Another challenge in the many-core domain is concerning the scalability of the algorithm itself. The problem of finding an optimal task mapping that reduces the peak or average temperature is an NP-hard problem. This problem grows by several orders of magnitude for many-core architecture. Moreover, specific optimization technique also need to be adapted to address the scalability issues. As an example, for the machine learning-based thermal management, the centralized learning model will become a bottleneck for many-core architectures. One possible direction to solve this is to have distributed learning agents. The associated challenges involve efficient handshaking of the agents and sharing the learning tables among multiple agents. Building a complete system to address the scalability problem is interesting, but challenging.

Scope: There are two aspects where the existing research works are lacking – heterogeneity of cores and thermal measurement. Most works on thermal management have focused on homogeneous cores. However, to extract performance and energy benefits, multiprocessor systems are equipped with heterogeneous cores. Examples include ARMs big.LITTLE architecture. The thermal behavior of different cores are different, even under the same workload. Additionally, not every thread or task can be mapped to every core. These challenges need to be factored in run-time thermal management policies. From the thermal measurements point, all the existing approaches have limitations – high simulation time for *HotSpot*, limited accuracy for *thermal gun* and recording speed for *thermal sensors*. One way to address the problem is to estimate the temperature from CPU performance statistics. Although some studies have been performed recently, accuracy is still an open problem.

Bibliography

[1] R. Yung, S. Rusu, and K. Shoemaker, "Future Trend of Microprocessor Design," in *Proceedings of the European Solid-State Circuits Conference (ESSCIRC)*. IEEE, 2002, pp. 43–46.

[2] B. Ackland, A. Anesko, D. Brinthaupt, S. Daubert, A. Kalavade, J. Knobloch, E. Micca, M. Moturi, C. Nicol, J. O'Neill, J. Othmer, E. Sackinger, K. Singh, J. Sweet, C. Terman, and J. Williams, "A Single-Chip, 1.6-Billion, 16-b MAC/s Multiprocessor DSP," *IEEE Journal of Solid-State Circuits,*, vol. 35, no. 3, pp. 412–424, 2000.

[3] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter Variations and Impact on Circuits and Microarchitecture," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2003, pp. 338–342.

[4] 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, no. 5, pp. 256–268, 1974.

[5] E. Karl, Y. Wang, Y.-G. Ng, Z. Guo, F. Hamzaoglu, U. Bhattacharya, K. Zhang, K. Mistry, and M. Bohr, "A 4.6GHz 162Mb SRAM Design in 22nm Tri-Gate CMOS Technology with Integrated Active VMIN-Enhancing Assist Circuitry," in *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2012, pp. 230–232.

[6] T. Ghani, M. Armstrong, C. Auth, M. Bost, P. Charvat, G. Glass, T. Hoffmann, K. Johnson, C. Kenyon, J. Klaus, B. McIntyre, K. Mistry, A. Murthy, J. Sandford, M. Silberstein, S. Sivakumar, P. Smith, K. Zawadzki, S. Thompson, and M. Bohr, "A 90nm High Volume Manufacturing Logic Technology Featuring Novel 45nm Gate Length Strained Silicon CMOS Transistors," in *IEEE International Electron Devices Meeting (IEDM)*, 2003, pp. 11.6.1–11.6.3.

[7] P. Packan, S. Akbar, M. Armstrong, D. Bergstrom, M. Brazier, H. Deshpande, K. Dev, G. Ding, T. Ghani, O. Golonzka, W. Han, J. He, R. Heussner, R. James, J. Jopling, C. Kenyon, S.-H. Lee, M. Liu, S. Lodha, B. Mattis, A. Murthy, L. Neiberg, J. Neirynck, S. Pae, C. Parker, L. Pipes, J. Sebastian, J. Seiple, B. Sell, A. Sharma, S. Sivakumar, B. Song, A. St.Amour, K. Tone, T. Troeger, C. Weber, K. Zhang, Y. Luo, and S. Natarajan, "High Performance 32nm Logic Technology Featuring 2nd Generation High-k + Metal Gate Transistors," in *IEEE International Electron Devices Meeting (IEDM)*, 2009, pp. 1–4.

[8] "International Technology Roadmap for Semiconductors (ITRS)," *Semiconductor Industry Association*, 2008. [Online]. Available: http://www.itrs.net

[9] S. Gupta, V. Moroz, L. Smith, Q. Lu, and K. Saraswat, "A Group IV Solution for 7 nm FinFET CMOS: Stress Engineering Using Si, Ge and Sn," in *IEEE International Electron Devices Meeting (IEDM)*, 2013, pp. 26.3.1–26.3.4.

[10] "SheevaPlug SoC," *Marvell Semiconductor*, 2009. [Online]. Available: http://www.marvell.com/solutions/plug-computers/

[11] "Cloverview Family of Intel SoC," *Intel Corporation*, 2009. [Online]. Available: http://ark.intel.com/products/codename/53606/cloverview

[12] S. Furber, D. Lester, L. Plana, J. Garside, E. Painkras, S. Temple, and A. Brown, "Overview of the SpiNNaker System Architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[13] J. De Oliveira and H. Van Antwerpen, "The Philips Nexperia Digital Video Platform," *Winning the SoC Revolution*, pp. 67–96, 2003.

[14] P. Cumming, "The TI OMAP Platform Approach to SoC," *Winning the SOC Revolution*, 2003.

[15] A. Artieri, V. Alto, R. Chesson, M. Hopkins, and M. Rossi, "Nomadik Open Multimedia Platform for Next-Generation Mobile Devices," *STMicroelectronics Technical Article TA305*, 2003.

[16] Y. Nishimichi, N. Higaki, M. Osaka, S. Horii, and H. Yoshida, "UniPhier: Series Development and SoC Management," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, ser. ASP-DAC '09. IEEE Press, 2009, pp. 540–545.

[17] J. Mottin, M. Cartron, and G. Urlini, "The STHORM Platform," in *Smart Multicore Embedded Systems*, M. Torquati, K. Bertels, S. Karlsson, and F. Pacull, Eds. Springer New York, 2014, pp. 35–43. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-8800-2_3

[18] "TILE-Gx8072 Processor Specification," *Tilera Corporation*, 2014. [Online]. Available: http://www.tilera.com/sites/default/files/images/products/TILE-Gx8072_PB041-03_WEB.pdf

[19] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[20] Goossens, K. and Dielissen, J. and Radulescu, A., "Æthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Design Test of Computers*, vol. 22, no. 5, pp. 414–421, Sept 2005.

[21] F. Moraes, N. Calazans, A. Mello, L. Mller, and L. Ost, "HERMES: An Infrastructure for Low Area Overhead Packet-Switching Networks on Chip," *Integration, the VLSI Journal*, vol. 38, no. 1, pp. 69 – 93, 2004.

[22] P. Stravers and J. Hoogerbrugge, "Homogeneous Multiprocessing and the Future of Silicon Design Paradigms," in *Proceedings of the International Symposium on VLSI Technology, Systems, and Applications*, 2001, pp. 184–187.

[23] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Perfor-

mance," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 2004, pp. 64–75.

[24] "Samsung Exynos 5 Octa," *Samsung Electronics*, 2014. [Online]. Available: www.samsung.com/exynos

[25] P. Greenhalgh, "big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," *ARM White Paper*, 2011. [Online]. Available: http://www.arm.com/products/processors/technologies/biglittleprocessing.php

[26] M. A. Watkins and D. H. Albonesi, "ReMAP: A Reconfigurable Heterogeneous Multicore Architecture," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, 2010, pp. 497–508.

[27] L. Chen and T. Mitra, "Shared Reconfigurable Fabric for Multi-core Customization," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2011, pp. 830–835.

[28] L. Jiashu, A. Das, and A. Kumar, "A Design Flow for Partially Reconfigurable Heterogeneous Multi-processor Platforms," in *Proceedings of the International Symposium on Rapid System Prototyping (RSP)*. IEEE, 2012, pp. 170–176.

[29] M. Santarini, "Zynq-7000 EPP Sets Stage for New Era of Innovations," *Xcell journal*, vol. 75, pp. 8–13, 2011.

[30] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level Design: Orthogonalization of Concerns and Platform-Based Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 19, no. 12, pp. 1523–1543, 2000.

[31] A. Pinto, A. Bonivento, A. L. Sangiovanni-Vincentelli, R. Passerone, and M. Sgroi, "System Level Design Paradigms: Platform-based Design and Communication Synthesis," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2004, pp. 537–563.

[32] K. Popovici, X. Guerin, F. Rousseau, P. S. Paolucci, and A. A. Jerraya, "Platform-based Software Design Flow for Heterogeneous MPSoC," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 4, pp. 39:1–39:23, 2008.

[33] W. Wolf, "Hardware-Software Co-Design of Embedded Systems," *Proceedings of the IEEE*, vol. 82, no. 7, pp. 967–989, 1994.

[34] J. Teich, "Hardware/Software Codesign: The Past, the Present, and Predicting the Future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1411–1430, May 2012.

[35] S. Borkar, "Design Perspectives on 22Nm CMOS and Beyond," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2009, pp. 93–94.

[36] L. Benini, A. Macii, E. Macii, and M. Poncino, "Selective Instruction Compression for Memory Energy Reduction in Embedded Systems," in *Proceedings of the ACM/IEEE international symposium on Low power electronics and design (ISLPED)*. ACM, 1999, pp. 206–211.

[37] T. Burd and R. Brodersen, "Energy Efficient CMOS Microprocessor Design," in *Proceedings of the International Conference on System Sciences*, vol. 1, 1995, pp. 288–297 vol.1.

[38] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Cosynthesis of Energy-Efficient Multimode Embedded Systems With Consideration of Mode-Execution Probabilities," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 24, no. 2, pp. 153–169, 2005.

[39] A. K. Singh, A. Das, and A. Kumar, "Energy Optimization by Exploiting Execution Slacks in Streaming Applications on Multiprocessor Systems," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2013, pp. 115:1–115:7.

[40] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Throughput-constrained DVFS for Scenario-Aware Dataflow Graphs," in *Proceedings of the IEEE Symposium on Real-Time and Embedded Technology and Applications (RTAS)*, 2013, pp. 175–184.

[41] J. Hu and R. Marculescu, "Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures Under Real-Time Constraints," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. IEEE Computer Society, 2004, pp. 10 234–.

[42] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms," *Journal of Systems Architecture*, vol. 56, no. 7, pp. 242 – 255, 2010.

[43] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw, and D. Sylvester, "Bubble Razor: Eliminating Timing Margins in an ARM Cortex-M3 Processor in 45 nm CMOS Using Architecturally Independent Error Detection and Correction," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 66–81, 2013.

[44] P. K. Saraswat, P. Pop, and J. Madsen, "Task Mapping and Bandwidth Reservation for Mixed Hard/Soft Fault-Tolerant Embedded Systems," in *Proceedings of the IEEE Symposium on Real-Time and Embedded Technology and Applications (RTAS)*. IEEE Computer Society, 2010, pp. 89–98.

[45] M. Prvulovic, Z. Zhang, and J. Torrellas, "ReVive: Cost-effective Architectural Support for Rollback Recovery in Shared-memory Multiprocessors," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 2002, pp. 111–122.

[46] C. Bolchini and A. Miele, "Reliability-Driven System-Level Synthesis for Mixed-Critical Embedded Systems," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2489–2502, 2013.

[47] A. Das, A. Kumar, and B. Veeravalli, "Communication and Migration Energy Aware Design Space Exploration for Multicore Systems with Intermittent Faults," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. European Design and Automation Association, 2013, pp. 1631–1636.

[48] I. Koren and C. Krishna, *Fault-tolerant Systems*. Morgan Kaufmann, 2007.

[49] F. Reynolds, "Thermally Accelerated Aging of Semiconductor Components," *Proceedings of the IEEE*, vol. 62, no. 2, pp. 212–222, 1974.

[50] R. Moazzami, J. C. Lee, and C. Hu, "Temperature Acceleration of Time-Dependent Dielectric Breakdown," *IEEE Transactions on Electron Devices*, vol. 36, no. 11, pp. 2462–2465, 1989.

[51] T. Brozek, Y. D. Chan, and C. R. Viswanathan, "Temperature Accelerated Gate Oxide Degradation Under Plasma-Induced Charging," *IEEE Electron Device Letters*, vol. 17, no. 6, pp. 288–290, 1996.

[52] S. Kothawade, K. Chakraborty, S. Roy, and Y. Han, "Analysis of Intermittent Timing Fault Vulnerability," *Microelectronics Reliability*, vol. 52, no. 7, pp. 1515 – 1522, 2012.

[53] P. M. Wells, K. Chakraborty, and G. S. Sohi, "Adapting to Intermittent Faults in Future Multicore Systems," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT)*. IEEE Computer Society, 2007, pp. 431–.

[54] S. Kumar, C. Kim, and S. Sapatnekar, "Adaptive Techniques for Overcoming Performance Degradation Due to Aging in CMOS Circuits," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 19, no. 4, pp. 603–614, 2011.

[55] S. Ramey, A. Ashutosh, C. Auth, J. Clifford, M. Hattendorf, J. Hicks, R. James, A. Rahman, V. Sharma, A. St.Amour, and C. Wiegand, "Intrinsic Transistor Reliability Improvements from 22nm Tri-Gate Technology," in *IEEE International Reliability Physics Symposium (IRPS)*, 2013, pp. 4C.5.1–4C.5.5.

[56] Y. Leblebici, "Design considerations for cmos digital circuits with improved hot-carrier reliability," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 7, pp. 1014–1024, 1996.

[57] V. Huard, C. Parthasarathy, A. Bravaix, C. Guerin, and E. Pion, "CMOS Device Design-in Reliability Approach in Advanced Nodes," in *IEEE International Reliability Physics Symposium*, 2009, pp. 624–633.

[58] A. Tiwari and J. Torrellas, "Facelift: Hiding and Slowing Down Aging in Multicores," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, 2008, pp. 129–140.

[59] F. Oboril, F. Firouzi, S. Kiamehr, and M. Tahoori, "Reducing NBTI-induced Processor Wearout by Exploiting the Timing Slack of Instructions," in *Proceed-*

ings of the Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), ser. CODES+ISSS '12. ACM, 2012, pp. 443–452.

[60] F. Ahmed, M. Sabry, D. Atienza, and L. Milor, "Wearout-Aware Compiler-Directed Register Assignment for Embedded Systems," in *Proceedings of the International Symposium on Quality Electronic Design (ISQED)*, 2012, pp. 33–40.

[61] L. Huang, F. Yuan, and Q. Xu, "Lifetime Reliability-aware Task Allocation and Scheduling for MPSoC Platforms," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. European Design and Automation Association, 2009, pp. 51–56.

[62] A. Agarwal, D. Blaauw, and V. Zolotov, "Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations," in *Proceedings of the International Conference on Computer Aided Design (ICCAD)*. IEEE Computer Society, 2003.

[63] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 21, no. 1, pp. 3–13, 2008.

[64] K. Bowman, S. Duvall, and J. Meindl, "Impact of Die-to-Die and Within-Die Parameter Fluctuations on the Maximum Clock Frequency Distribution for Gigascale Integration," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 2, pp. 183–190, 2002.

[65] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-aware Task Allocation and Scheduling for MPSoC," in *Proceedings of the International Conference on Computer Aided Design (ICCAD)*. IEEE Press, 2007, pp. 598–603.

[66] L. Singhal and E. Bozorgzadeh, "Process variation aware system-level task allocation using stochastic ordering of delay distributions," in *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, ser. ICCAD '08. IEEE Press, 2008, pp. 570–574.

[67] F. Paterna, A. Acquaviva, A. Caprara, F. Papariello, G. Desoli, and L. Benini, "Variability-Aware Task Allocation for Energy-Efficient Quality of Service Pro-

visioning in Embedded Streaming Multimedia Applications," *Computers, IEEE Transactions on*, vol. 61, no. 7, pp. 939–953, 2012.

[68] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-Aware Microarchitecture: Modeling and Implementation," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 1, no. 1, pp. 94–125, 2004.

[69] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[70] A. Kumar, B. Mesman, H. Corporaal, and Y. Ha, "Iterative Probabilistic Performance Prediction for Multi-Application Multiprocessor Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 4, pp. 538–551, 2010.

[71] A. Das, A. Kumar, and B. Veeravalli, "Temperature Aware Energy-Reliability Trade-offs for Mapping of Throughput-Constrained Applications on Multimedia MPSoCs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. European Design and Automation Association, 2014.

[72] A. Das, A. Kumar, and B. Veeravalli, "Reliability-driven Task Mapping for Lifetime Extension of Networks-on-chip Based Multiprocessor Systems," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. European Design and Automation Association, 2013, pp. 689–694.

[73] A. Das, A. Kumar, and B. Veeravalli, "Aging-aware Hardware-software Task Partitioning for Reliable Reconfigurable Multiprocessor Systems," in *Proceedings of the International Conference on Compilers, Architecturesand Synthesis for Embedded Systems (CASES)*. IEEE Press, 2013, pp. 1:1–1:10.

[74] A. Das, A. Kumar, and B. Veeravalli, "Energy-Aware Communication and Remapping of Tasks for Reliable Multimedia Multiprocessor Systems," in *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE Computer Society, 2012, pp. 564–571.

[75] A. Das and A. Kumar, "Fault-aware Task Re-mapping for Throughput Constrained Multimedia Applications on NoC-based MPSoCs," in *Proceedings of the International Symposium on Rapid System Prototyping (RSP)*. IEEE 2012, pp. 149–155.

[76] A. Das, A. Kumar, and B. Veeravalli, "Communication and Migration Energy Aware Task Mapping for Reliable Multiprocessor Systems," *Elsevier Future Generation Computer Systems*, vol. 30, no. 0, pp. 216 – 228, 2014.

[77] A. Das, A. Kumar, and B. Veeravalli, "Energy-aware Task Mapping and Scheduling for Reliable Embedded Computing Systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 2s, pp. 72:1–72:27, 2014.

[78] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2014.

[79] S. Stuijk, "Predictable Mapping of Streaming Applications on Multiprocessors," *PhD Thesis, Eindhoven University of Technology*, 2007.

[80] S. Sriram and S. Bhattacharyya, *Embedded Multiprocessors; Scheduling and Synchronization*. Marcel Dekker, 2000.

[81] S. Stuijk, M. Geilen, and T. Basten, "Exploring Trade-offs in Buffer Requirements and Throughput Constraints for Synchronous Dataflow Graphs," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2006, pp. 899–904.

[82] A. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, and M. Mousavi, "Throughput analysis of synchronous data flow graphs," in *Proceedings of the International Conference on Application of Concurrency to System Design (ACSD)*, vol. 6. IEEE Computer Society, 2006, pp. 25–36.

[83] J. R. Black, "Electromigration Failure Modes in Aluminum Metallization for Semiconductor Devices," *Proceedings of the IEEE*, vol. 57, no. 9, pp. 1587–1594, 1969.

[84] J. S. S. T. Association, "Failure Mechanisms and Models for Semiconductor Devices," *JEDEC Publication JEP122-B*, 2003.

[85] T. Yamamoto, K. Uwasawa, and T. Mogami, "Bias Temperature Instability in Scaled p+ Polysilicon Gate p-MOSFET's," *IEEE Transactions on Electron Devices*, vol. 46, no. 5, pp. 921–926, 1999.

[86] M. Alam, "A Critical Examination of the Mechanics of Dynamic NBTI for PMOSFETs," in *IEEE International Electron Devices Meeting (IEDM)*, 2003, pp. 14.4.1–14.4.4.

[87] M. Alam and S. Mahapatra, "A Comprehensive Model of {PMOS} {NBTI} Degradation," *Microelectronics Reliability*, vol. 45, no. 1, pp. 71 – 81, 2005.

[88] E. Takeda and N. Suzuki, "An Empirical Model for Device Degradation Due to Hot-Carrier Injection," *IEEE Electron Device Letters*, vol. 4, no. 4, pp. 111–113, 1983.

[89] S. Tam, P.-K. Ko, and C. Hu, "Lucky-Electron Model of Channel Hot-Electron Injection in MOSFET'S," *IEEE Transactions on Electron Devices*, vol. 31, no. 9, pp. 1116–1125, 1984.

[90] S. Downing and D. Socie, "Simple Rainflow Counting Algorithms ," *International Journal of Fatigue*, vol. 4, no. 1, pp. 31 – 40, 1982.

[91] V. Radhakrishnan, "On the bilinearity of the coffin-manson low-cycle fatigue relationship," *International Journal of Fatigue*, vol. 14, no. 5, pp. 305 – 311, 1992.

[92] T. Shimokawa and S. Tanaka, "A statistical consideration of miner's rule," *International Journal of Fatigue*, vol. 2, no. 4, pp. 165 – 170, 1980.

[93] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and L. Shang, "System-level Reliability Modeling for MPSoCs," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2010, pp. 297–306.

[94] K. Chopra, C. Zhuo, D. Blaauw, and D. Sylvester, "A Statistical Approach for Full-chip Gate-oxide Reliability Analysis," in *Proceedings of the International Conference on Computer Aided Design (ICCAD)*. IEEE Press, 2008, pp. 698–705.

[95] L. Huang, F. Yuan, and Q. Xu, "On Task Allocation and Scheduling for Lifetime Extension of Platform-Based MPSoC Designs," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 22, no. 12, pp. 2088–2099, 2011.

[96] T. Chantem, X. Hu, and R. Dick, "Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 19, no. 10, pp. 1884–1897, 2011.

[97] I. Ukhov, M. Bao, P. Eles, and Z. Peng, "Steady-state Dynamic Temperature Analysis and Reliability Optimization for Embedded Multiprocessor Systems," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2012, pp. 197–204.

[98] Z. Gu, C. Zhu, L. Shang, and R. Dick, "Application-Specific MPSoC Reliability Optimization," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 16, no. 5, pp. 603–608, 2008.

[99] L. Huang and Q. Xu, "Lifetime Reliability for Load-Sharing Redundant Systems With Arbitrary Failure Distributions," *IEEE Transactions on Reliability*, vol. 59, no. 2, pp. 319–330, 2010.

[100] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The Case for Lifetime Reliability-Aware Microprocessors," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 2004, pp. 276–287.

[101] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Thermal-Aware Task Allocation and Scheduling for Embedded Systems," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. IEEE Computer Society, 2005, pp. 898–899.

[102] J. Cui and D. Maskell, "A Fast High-Level Event-Driven Thermal Estimator for Dynamic Thermal Aware Scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 31, no. 6, pp. 904–917, 2012.

[103] D. Rai, H. Yang, I. Bacivarov, J.-J. Chen, and L. Thiele, "Worst-case Temperature Analysis for Real-time Systems," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2011, pp. 1–6.

[104] B. H. Meyer, A. S. Hartman, and D. E. Thomas, "Cost-effective Slack Allocation for Lifetime Improvement in NoC-based MPSoCs," in *Proceedings of the Confer-*

ence on Design, Automation and Test in Europe (DATE). European Design and Automation Association, 2010, pp. 1596–1601.

[105] A. S. Hartman, D. E. Thomas, and B. H. Meyer, "A Case for Lifetime-aware Task Mapping in Embedded Chip Multiprocessors," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS).* ACM, 2010, pp. 145–154.

[106] L. Huang and Q. Xu, "Energy-efficient Task Allocation and Scheduling for Multi-mode MPSoCs Under Lifetime Reliability Constraint," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE).* European Design and Automation Association, 2010, pp. 1584–1589.

[107] C.-L. Chou and R. Marculescu, "FARM: Fault-aware Resource Management in NoC-based Multiprocessor Platforms," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE).* European Design and Automation Association, 2011, pp. 1 –6.

[108] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined DVFS and Mapping Exploration for Lifetime and Soft-Error Susceptibility Improvement in MPSoCs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE).* European Design and Automation Association, 2014.

[109] B. Dave and N. Jha, "COFTA: Hardware-Software Co-Synthesis of Heterogeneous Distributed Embedded Systems for Low Overhead Fault Tolerance," *IEEE Transactions on Computers*, vol. 48, no. 4, pp. 417–441, 1999.

[110] Y. Xie, L. Li, M. Kandemir, N. Vijaykrishnan, and M. Irwin, "Reliability-aware Co-synthesis for Embedded Systems," *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 49, no. 1, pp. 87–99, 2007.

[111] C. Bolchini, A. Miele, F. Salice, D. Sciuto, and L. Pomante, "Reliable System Co-Design: the FIR Case Study," in *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct 2004, pp. 433–441.

[112] P. v. Stralen and A. Pimentel, "A SAFE Approach Towards Early Design Space Exploration of Fault-tolerant Multimedia MPSoCs," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2012, pp. 393–402.

[113] V. Izosimov, I. Polian, P. Pop, P. Eles, and Z. Peng, "Analysis and Optimization of Fault-tolerant Embedded Systems with Hardened Processors," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. European Design and Automation Association, 2009, pp. 682–687.

[114] C. Bolchini and A. Miele, "Reliability-Driven System-Level Synthesis of Embedded Systems," in *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct 2010, pp. 35–43.

[115] M. Glaß, M. Lukasiewycz, T. Streichert, C. Haubelt, and J. Teich, "Reliability-aware System Synthesis," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. EDA Consortium, 2007, pp. 409–414.

[116] C. Zhu, Z. P. Gu, R. P. Dick, and L. Shang, "Reliable Multiprocessor System-on-chip Synthesis," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2007, pp. 239–244.

[117] A. Jhumka, S. Klaus, and S. A. Huss, "A Dependability-Driven System-Level Design Approach for Embedded Systems," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. IEEE Computer Society, 2005, pp. 372–377.

[118] C. Yang and A. Orailoglu, "Predictable Execution Adaptivity Through Embedding Dynamic Reconfigurability into Static MPSoC Schedules," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2007, pp. 15–20.

[119] J. Huang, J. O. Blech, A. Raabe, C. Buckl, and A. Knoll, "Analysis and Optimization of Fault-tolerant Task Scheduling on Multiprocessor Embedded Systems," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2011, pp. 247–256.

[120] C. Lee, H. Kim, H.-w. Park, S. Kim, H. Oh, and S. Ha, "A Task Remapping Technique for Reliable Multi-core Embedded Systems," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2010, pp. 307–316.

[121] T. Wei, X. Chen, and S. Hu, "Reliability-Driven Energy-Efficient Task Scheduling for Multiprocessor Real-Time Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 30, no. 10, pp. 1569–1573, 2011.

[122] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge, "Reliability Modeling and Management in Dynamic Microprocessor-based Systems," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2006, pp. 1057–1060.

[123] V. Hanumaiah and S. Vrudhula, "Temperature-Aware DVFS for Hard Real-Time Applications on Multicore Processors," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1484–1494, 2012.

[124] M. A. Faruque, J. Jahn, and J. Henkel, "Runtime Thermal Management Using Software Agents for Multi- and Many-Core Architectures," *IEEE Design & Test of Computers*, vol. 27, no. 6, pp. 58–68, 2010.

[125] A. S. Hartman and D. E. Thomas, "Lifetime Improvement Through Runtime Wear-based Task Mapping," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2012, pp. 13–22.

[126] T. Chantem, Y. Xiang, X. S. Hu, and R. P. Dick, "Enhancing Multicore Reliability Through Wear Compensation in Online Assignment and Scheduling," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. European Design and Automation Association, 2013, pp. 1373–1378.

[127] A. Coskun, T. Rosing, and K. Gross, "Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 28, no. 10, pp. 1503–1516, 2009.

[128] R. Z. Ayoub and T. S. Rosing, "Predict and Act: Dynamic Thermal Management for Multi-core Processors," in *Proceedings of the ACM/IEEE international symposium on Low power electronics and design (ISLPED)*. ACM, 2009, pp. 99–104.

[129] R. Cochran and S. Reda, "Consistent Runtime Thermal Prediction and Control Through Workload Phase Detection," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2010, pp. 62–67.

[130] F. Sironi, M. Maggio, R. Cattaneo, G. Del Nero, D. Sciuto, and M. Santambrogio, "ThermOS: System Support for Dynamic Thermal Management of Chip Multi-Processors," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2013, pp. 41–50.

[131] C. Bolchini, M. Carminati, A. Miele, A. Das, A. Kumar, and B. Veeravalli, "Run-time mapping for reliable many-cores based on energy/performance trade-offs," in *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2013, pp. 58–64.

[132] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini, "Workload and User Experience-aware Dynamic Reliability Management in Multicore Processors," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2013, pp. 2:1–2:6.

[133] W. Lee, K. Patel, and M. Pedram, "GOP-level Dynamic Thermal Management in MPEG-2 Decoding," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 16, no. 6, pp. 662–672, 2008.

[134] R. Jayaseelan and T. Mitra, "Dynamic Thermal Management via Architectural Adaptation," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2009, pp. 484–489.

[135] Y. Ge and Q. Qiu, "Dynamic Thermal Management for Multimedia Applications Using Machine Learning," in *Proceeding of the Annual Design Automation Conference (DAC)*. ACM, 2011, pp. 95–100.

[136] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic Learning for Thermal-aware Power Budgeting in Many-core Architectures," in *Proceedings of the Conference on*

Hardware/Software Codesign and System Synthesis (CODES+ISSS). ACM, 2011, pp. 189–196.

[137] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems," in *Proceeding of the Annual Design Automation Conference (DAC)*, 2014.

[138] S. Stuijk, M. Geilen, and T. Basten, "SDF3: SDF For Free," in *Proceedings of the International Conference on Application of Concurrency to System Design (ACSD)*. IEEE Computer Society, 2006, pp. 276–278.

[139] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor, "Concepts and Implementation of Spatial Division Multiplexing for Guaranteed Throughput in Networks-on-Chip," *IEEE Transactions on Computers*, vol. 57, no. 9, pp. 1182–1195, 2008.

[140] W. Liao, L. He, and K. Lepak, "Temperature and Supply Voltage Aware Performance and Power Modeling at Microarchitecture Level," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 24, no. 7, pp. 1042–1053, 2005.

[141] M. Bao, A. Andrei, P. Eles, and Z. Peng, "Temperature-aware Idle Time Distribution for Energy Optimization with Dynamic Voltage Scaling," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. European Design and Automation Association, 2010, pp. 21–26.

[142] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, "NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 16, no. 2, pp. 113–129, 2005.

[143] G. Coley, "Beagleboard System Reference Manual," *BeagleBoard. org*, p. 81, 2009.

[144] W. Zhao and Y. Cao, "Predictive Technology Model for nano-CMOS Design Exploration," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 3, no. 1, 2007.

[145] S. K. Reinhardt and S. S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2000, pp. 25–36.

[146] P. Axer, M. Sebastian, and R. Ernst, "Reliability Analysis for MPSoCs with Mixed-critical, Hard Real-time Constraints," in *Proceedings of the Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2011, pp. 149–158.

[147] S.-W. Kwak, B.-J. Choi, and B.-K. Kim, "An Optimal Checkpointing-Strategy for Real-Time Control Systems Under Transient Faults," *IEEE Transactions on Reliability*, vol. 50, no. 3, pp. 293–301, Sep 2001.

[148] C. Krishna and A. Singh, "Reliability of Checkpointed Real-Time Systems Using Time Redundancy," *IEEE Transactions on Reliability*, vol. 42, no. 3, pp. 427–435, Sep 1993.

[149] L. Goh, B. Veeravalli, and S. Viswanathan, "Design of Fast and Efficient Energy-aware Gradient-based Scheduling Algorithms Heterogeneous Embedded Multiprocessor Systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 20, no. 1, pp. 1–12, 2009.

[150] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 13, no. 3, pp. 260–274, 2002.

[151] Y.-K. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406–471, 1999.

[152] J. Blazewicz, "Scheduling Dependent Tasks with Different Arrival Times to Meet Deadlines," in *Proceedings of the International Workshop organized by the Commision of the European Communities on Modelling and Performance Evaluation of Computer Systems*. North-Holland Publishing Co., 1976, pp. 57–65.

[153] R. Dick. (2013) Embedded System Synthesis Benchmarks Suite (E3S).

[154] M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes, "The ALPBench benchmark suite for complex multimedia applications," in *Workload Characterization Symposium.* IEEE, 2005, pp. 34–45.

[155] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[156] "Perf: Linux Profiling with Performance Counters," 2012. [Online]. Available: https://perf.wiki.kernel.org

[157] J. Treibig, G. Hager, and G. Wellein, "LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments," in *International Conference on Parallel Processing Workshops (ICPPW)*, 2010, pp. 207–216.

[158] V. Pallipadi and A. Starikovskiy, "The Ondemand Governor," in *Proceedings of the Linux Symposium*, vol. 2. sn, 2006, pp. 215–230.

List of Publications

Journals

- Anup Das, Akash Kumar and Bharadwaj Veeravalli. "Energy-aware Task Mapping and Scheduling for Reliable Embedded Computing Systems". In *Transactions on Embedded Computing Systems (TECS)*, vol. 13, pp. 72:1-72:27. ACM, 2014. doi:10.1145/2544375.2544392.

- Anup Das, Akash Kumar and Bharadwaj Veeravalli. "Communication and Migration Energy Aware Task Mapping for Reliable Multiprocessor Systems". In *Future Generation Computer Systems (FGCS)*, vol. 30, pp. 216-228. Elsevier, 2014. doi: 10.1016/j.future.2013.06.016.

- Anup Das, Akash Kumar and Bharadwaj Veeravalli. "Reliability-Aware Hardware-Software Co-Design of Reconfigurable Multiprocessor Systems". In *Transactions on Computers (TC)* (under review). IEEE 2014.

- Anup Das, Akash Kumar and Bharadwaj Veeravalli. "Reliability and Energy-Aware Mapping and Scheduling of Multimedia Applications on Multiprocessor Systems". In *Transactions on Parallel and Distributed Systems (TPDS)* (under review). IEEE 2014.

Conferences

- Anup Das, Rishad A. Shafik, Geoff V. Merrett, Bashir M. Al-Hashimi, Akash Kumar and Bharadwaj Veeravalli. "Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems". In *Design Automation Conference (DAC)*. ACM/IEEE, 2014.

- Anup Das, Akash Kumar and Bharadwaj Veeravalli. "Temperature Aware Energy-Reliability Trade-offs for Mapping of Throughput-Constrained Applications on Multimedia MPSoCs". In *Conference on Design Automation and Test in Europe (DATE)*, pp. 1-6. IEEE/ACM, 2014. doi: 10.7873/DATE2014.115.

- Anup Das, Akash Kumar, Bharadwaj Veeravalli, Cristiana Bolchini and Antonio Miele. "Combined DVFS and Mapping Exploration for Lifetime and Soft-Error Susceptibility Improvement in MPSoCs". In *Conference on Design Automation and Test in Europe (DATE)*, pp. 1-6. IEEE/ACM, 2014. doi: 10.7873/DATE2014.074.

- Anup Das, Akash Kumar and Bharadwaj Veeravalli. "Aging-aware Hardware-Software Task Partitioning for Reliable Reconfigurable Multiprocessor Systems". In *Conference on Compiler, Architecture and Synthesis for Embedded Systems (CASES)*, pp. 1-10. IEEE/ACM, 2013. doi: 10.1109/CASES.2013.6662505.

- Cristiana Bolchini, Matteo Carminati, Antonio Miele, Anup Das, Akash Kumar and Bharadwaj Veeravalli. "Run-time Mapping for Reliable Many-cores Based on Energy/Performance Trade-offs". In *Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 58-64. IEEE, 2013. doi: 10.1109/DFT.2013.6653583.

- Amit Kumar Singh, Anup Das and Akash Kumar. "Energy Optimization by Exploiting Execution Slacks in Streaming Applications on Multiprocessor Systems". In *Design Automation Conference (DAC)*, pp. 1-7. ACM/IEEE, 2013. doi: 10.1145/2463209.2488875.

- Anup Das, Akash Kumar and Bharadwaj Veeravalli. "Reliability-Driven Task Mapping for Lifetime Extension of Networks-on-Chip Based Multiprocessor Systems". In *Conference on Design Automation and Test in Europe (DATE)*, pp. 689-694. IEEE/ACM, 2013. doi: 10.7873/DATE.2013.149.

- Anup Das, Akash Kumar and Bharadwaj Veeravalli. "Communication and Migration Energy Aware Design Space Exploration for Multicore Systems with Intermittent Faults". In *Conference on Design Automation and Test in Europe (DATE)*, pp. 1631-1636. IEEE/ACM, 2013. doi: 10.7873/DATE.2013.331.

- Anup Das, Akash Kumar and Bharadwaj Veeravalli. "Energy-Aware Communication and Remapping of Tasks for Reliable Multimedia Multiprocessor Systems". In *International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 564-571. IEEE, 2012. doi: 10.1109/ICPADS.2012.82.

- Anup Das and Akash Kumar. "Fault-Aware Task Re-Mapping for Throughput Constrained Multimedia Applications on NoC-based MPSoC". In *Symposium on Rapid System Prototyping (RSP)*, pp. 149-155. IEEE, 2012. doi: 10.1109/RSP.2012.6380704.