2015

Polymorphic computing abstraction for heterogeneous architectures

Swamy Dhoss Ponpandi
Iowa State University

Polymorphic computing abstraction for heterogeneous architectures

by

Swamy Dhoss Ponpandi

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Akhilesh Tyagi, Major Professor
Soma Chaudhuri
Arun Somani
Zhao Zhang
Manimaran Govindarasu

Iowa State University

Ames, Iowa

2015

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to Professor Akhilesh Tyagi for the never ending ideas and thoughts which motivated the issues being addressed in this thesis. Prof. Tyagi has a unique way of viewing open problems and I have benefited much from his thought process. I hope to keep our discussions going. I am also very thankful for the many personal help from Prof. Tyagi over the years.

I would also like to thank Professor Arun Somani, Professor Soma Chaudhuri, Professor Manimaran Govindarasu and Professor Zhao Zhang for agreeing to serve as my PhD committee members and their valuable suggestions to improve this thesis is very much appreciated.

I would also like to thank my adviser and professors from South Dakota School of Mines and Technology. Professors Michael Batchelder, Larry Simonson, Brian Hemmelman, Timor Paltashev and Abul Hassan have provided great help during my years at School of Mines. I cannot forget the help of Carolyn Brich and we will miss her, RIP. My friends from Rapid City - Ravindra Kharde, Srikanth Utukuri, Prithvi will be happy to read this thesis. I appreciate friends from ISU who have shaped my technical knowledge. I would like to thank Veerendra Allada, Ka-Ming Keung and Jungmin Park for the good friendship.

My father, mother, brother and sister have been source of moral support and encouragement for which I am grateful. My wife and sons have supported me all the time and I am grateful for that too.

ABSTRACT

Integration of multiple computing paradigms onto system on chip (SoC) has pushed the boundaries of design space exploration for hardware architectures and computing system software stack. The heterogeneity of computing styles in SoC has created a new class of architectures referred to as *Heterogeneous Architectures*. Novel applications developed to exploit the different computing styles are *user centric* for embedded SoC. Software and hardware designers are faced with several challenges to harness the full potential of heterogeneous architectures. Applications have to execute on more than one compute style to increase overall SoC resource utilization. The implication of such an abstraction is that application threads need to be *polymorphic*. Operating system layer is thus faced with the problem of scheduling polymorphic threads. Resource allocation is also an important problem to be dealt by the OS. Morphism evolution of application threads is constrained by the availability of heterogeneous computing resources. Traditional design optimization goals such as computational power and lower energy per computation are inadequate to satisfy user centric application resource needs. Resource allocation decisions at application layer need to permeate to the architectural layer to avoid conflicting demands which may affect energy-delay characteristics of application threads. We propose *Polymorphic computing abstraction* as a unified computing model for heterogeneous architectures to address the above issues. Simulation environment for polymorphic applications is developed and evaluated under various scheduling strategies to determine the effectiveness of polymorphism abstraction on resource allocation. User satisfaction model is also developed to complement polymorphism and used for optimization of resource utilization at application and network layer of embedded systems.

CHAPTER 1. INTRODUCTION

Embedded systems are traditionally characterized by strict real time constraints, low power consumption and dedicated functionality. Modern definition of embedded system has expanded the characterization contour to include mobile computing systems. Examples of mobile computing systems include 1) content consumption devices such as smart phones, tablets and 2) emerging technology such as wearable computing products. Mobile computing systems use general purpose *multicore* chips, variants of *Linux* operating system and complex *user interaction* based interface. Mobile computing systems also have *slack* in meeting deadline constraints and tend to search for performance-power optimal solutions in the design space.

Multicore architectures [TI, Nvi] are the current hardware solution to mitigate challenges due to increasing wire delays and leakage current. These issues dominate VLSI design as the gate length of transistors approaches the diameter of atom in future technology nodes. The number of compute cores, which can vary in energy-delay profiles, integrated in multicore architectures is increasing with each product cycle to target computation needs of novel applications [EDFL13, LSMW11, SKPK10] in embedded systems. The implication of multicore architectures and novel applications is *non-trivial* for the problem of *design space* exploration. This thesis presents the concept of polymorphism, user satisfaction and experiments on network-on-chip model to explore the design space offered by multicore architectures. The next section motivates the idea of *Polymorphism* in the context of multicore architectures.

1.1 Polymorphism

Applications are abstracted as communicating threads primarily to share compute resources with other applications. Threads also increase resource utilization from operating system (OS)

perspective. However, threads are *statically* bound to a specific compute resource such as CPU, FPGA, GPU at application design phase. For example, a thread applying a smoothing filter to a video frame will be executed on CPU while another kernel thread which computes motion compensation vectors is bound to GPU during the design phase of video decoder application. The resources on multicore architecture will be under-utilized by such static allocation policies. The OS scheduler is denied the choice of *resource allocation* for a thread if such a thread can utilize more than one resource in the hardware platform at run time. The restriction of static binding can be loosened in such cases. The thread *morphs* itself into an available *morphism or avatar* to take advantage of dynamic set of resources in each scheduling cycle and hence, threads are *polymorphic*. In polymorphic context, morphisms are alternate ways of implementing the thread's function. A thread's morphism decides the resource consumption and its contribution to the application performance metric. The morphisms differ in their resource requirements to implement a thread's functionality in a given hardware platform. Hence, every possible morphism may not be supported across different platforms. For example, let us assume that a mobile internet browser thread has three morphisms. The first thread morphism (M_1) may be tuned for faster execution of all text and multimedia contents of the webpage, a second morphism (M_2) may be optimized for memory storage such as bandwidth while loading the page, while the third (M_3) may be designed to operate in a power-saver mode in which only text will be shown. Moreover, a threads behavior could be completely implemented in software. Multiple software implementations are possible for a thread by changing the algorithm used to realize its functionality, which is another way to realize morphisms.

The burden is on the OS scheduler to choose an appropriate morphism at *run time* for one or more ready to run threads depending on the optimization goal of interest. The scheduler has the flexibility to choose from the three morphisms (M_1, M_2, M_3) for the browser thread. M_1, M_2 and M_3 may have different resource requirements. Resource requirements need to be satisfied before a thread can morph into a specific morphism. For example, in the browser thread, M_1 may use 1 CPU for I/O related functions such as mouse click or text inputs, 1 CPU for loading/executing java applets, 2 FPGA tiles for multimedia content. Similarly, M_2 may use 3 CPUs - one for I/O related functions, one for loading/executing java applets and

one hyper-threading capable CPU for multimedia content. Finally, M_3 may use a low power CPU to execute load text only portion of the webpage. The availability of variegated computing resources is commonplace in contemporary embedded multicore architectures [TI, Nvi] and such architectures are the vanguard to achieve the projected performance doubling in current and future VLSI technology nodes. Polymorphic thread paradigm stitches piece-wise performance-energy profiles to explore multicore architecture design space. The browser thread can exist in M_1 , M_2 or M_3 morphism, each with different performance-energy profile, during the lifetime of the browser application. The convenience of choosing a morphism will be shown to result in better utilization of compute resources and hence, performance improvement in a multicore platform when compared with statically bound threads. The motivation for user satisfaction, which is an optimization goal for polymorphic thread paradigm based embedded system design, is presented next.

Existing optimization goals at the hardware layer in VLSI design target low level parameters such as power, delay or some function of these. Mature EDA tools optimize low level parameters for average case design points. Higher level abstractions (OS layer, APP layer) further focus on average case by being oblivious to varying resource allocation needs of threads. Such optimization metrics are usually monotonic in the sense that there is minimal a priori information about the application characteristics. Traditional computing systems such as webservers, personal computers which are characterized by low degree of user interaction have benefited from monotonous metrics used in the OS scheduler. However, the degree of user interaction in embedded system is very much different compared to traditional computing systems. A typical usage scenario of smartphone is short bursts of activity followed by periods of inactivity. The OS scheduler is under duress to choose morphism for one or more threads and use or free resources to satisfy the morphism resource requirements. The application's ability to continue execution is dependent on scheduling policies in such embedded systems. For example, an application will be terminated if it fails to respond in 5 seconds in iOS. Hence, the scheduler is on a short leash and should provide best effort resource allocation for the *user behavior* in that short time window. Traditional methods of allocating resources such as earliest deadline first, first come first serve will fall short to optimize a given design parameter in scenarios where

user behavior is not taken into account. Alternatively, the problem of resource allocation can be addressed by modeling the user perception of an application. Resources are only allocated to the application to satisfy the user perception and any resource allocated beyond this limit is not going to change the perceived *satisfaction* of the user due to the incremental resource allocation. Hence, the scheduler may use extra resource for other threads to further increase the *user satisfaction*. The next section describes the need for modeling of user satisfaction as an optimization function.

1.2 User satisfaction

Human sensory interfaces are naturally redundant. Redundant information is provided to the brain through one or more sensory interface such as visual system, auditory system, olfactory and other biological systems. Further, the range of sensory stimulus to which a given sensory interface responds is also limited. This is especially true and clearly observed for visual and auditory system. Human Audio/Visual System (HAVS) under certain conditions such as changing lighting effects takes finite time to adapt. This effect is taken into account for computer graphics generated scenes to give a realistic effect when a subject enters into a dark room from bright light. Perception is the model built by the brain to make sense of the information relayed by sensory interfaces. Consider, a human subject watching movie on a 1080p screen and 720p screen standing at a distance. The ability to resolve details in the image projected on the screen falls as the distance increases. At a specific distance from the screen, the subject will perceive both images to be identical which implies the subject cannot differentiate between the screen resolutions. This is due to the limitation of HAVS to resolve spatial frequencies. Under low lighting conditions, the ability of HAVS to detect color differences is poor but it is sensitive to changes in illumination. Again such differences in response arise due to the asymmetry in HAVS. The cone photoreceptors in HAVS is responsible for color vision and the lower number of these receptors limits the color sensitivity.

Applications interacting with HAVS and using compute resources to optimize design parameters are also limited by HAVS characteristics. The limitation arises due to the inability of HAVS to distinguish between two outputs of an application, O_1 and O_2, which use different

compute resources, R_1 and R_2. Hence, it may be advantageous to choose the application output which uses lower compute resources in situations such as energy optimization. For example, frame rate and resolution are the primary parameters of a video decoding application which decide the amount of resource consumed. Assuming a constant resolution, varying the frame rate gives different experiences for users depending on the nature of the application. The final interaction of a video decoding application is with HAVS which limits the user experience when measured as a function of frame rate. A video playback at 100 fps (frames per second) (O_1) is essentially indistinguishable from 30 fps (O_2) for most videos due to inability of HAVS to notice any significant variation in spatial frequency of the image. The compute resources (GPU, CPU - R_1) allocated to have video playback at 100 fps may be used to better display the video at 30 fps (CPU - R_1) with an anti-aliasing filter running on the GPU. This scenario will result in better user satisfaction rather than optimizing in one dimension (fps). Similarly, video playback below 15 fps will result in choppy video and audio synchronization problems. Reclaiming compute resources from the video application in this scenario will further degrade the user satisfaction. The user satisfaction in the region between 15 fps and 30 fps is user behavior dependent and is expected to be non-linear. This characteristic of HAVS which determines the user's perceived satisfaction is modeled using a *sigmoid* function. The resources are allocated to maximize user satisfaction in this work and the results are compared with existing thread scheduling policies. The next section introduces the experimental framework used in this thesis for design space exploration of multicore architectures.

1.3 Network on chip model

Multicore chips have two or more compute resource and interconnection network to facilitate communication between compute cores. Further, there are input/output cores, cache and other peripherals which may use the interconnection network. Multicore chips are customized to the target market by integrating various combinations of compute resource, memory, I/O and interconnection network into single chip systems referred to as systems-on-chip (SoC). SoCs have been widely used in embedded computing devices and are predicted to dominate handheld, tablet devices market. Early SoC architectures used few cores and other peripherals.

Hence, simple interconnection networks such as point to point, rings, ad hoc networks worked well to optimize the digital logic for maximum clock frequency. However, due to increasing wire delays below 22nm technology node, the previous digital design techniques for global clocking cannot be sustained. Further, scalability is limited in ad hoc, point to point, ring networks. Network-on-chip (NoC) is a communication abstraction model for various styles of interconnection networks.

NoC paradigm is based on *globally asynchronous locally synchronous* design principle which mitigates the effect of propagation delay of clock signal due to increasing global interconnect resistance. Asynchronous transfer of information between locally synchronous components in NOC decouples communication characteristics of the network from compute core design parameters. This allows for NoC components to be designed independently. Due to the asynchronous nature of communication in NoC, communicating entities (threads, CPUs etc.) are free to ignore global clock synchronization problems. NOC abstraction floor plan is illustrated in Figure 1.1. It is composed of two major abstractions - compute cores and interconnection network.

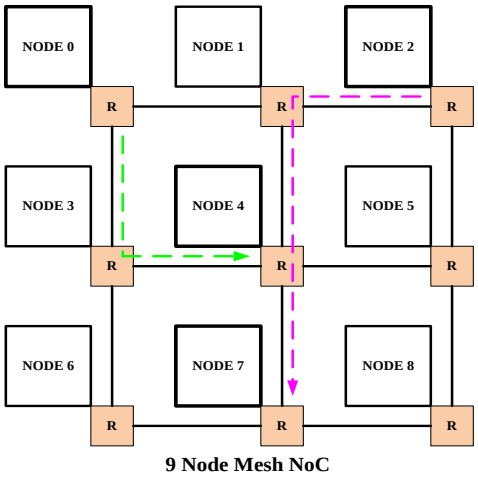

Figure 1.1: Generic Multicore Architecture

The properties of compute resources are abstracted by *nodes* of the NoC. Nodes can represent different computing styles such as CPU, FPGA or GPU. The interconnection network shown in Figure 1.1 is a mesh network. Nodes interface to the interconnection network through

router (**R** in Figure 1.1). Routers can receive and transmit data packets through the network. This enables communication between nodes. The internal architecture of router consists of buffers, arbiter, ports and control logic to perform the various functions needed by applications using the NoC model. Storing and forwarding data packets, assigning buffer resources to packets, granting port to winning data packets are some basic functions of router. *PolyNoC* is our implementation of a polymorphic NoC simulation environment capable of executing polymorphic applications. The simulation environment is parametrized to model different configurations of NoC. User satisfaction based scheduling is built into the scheduling layer of the PolyNoC software stack. The ideas introduced in Section 1.1 and Section 1.2 are evaluated using PolyNoC for audio and video benchmarks. Traditional scheduling policies are compared against the user satisfaction model based scheduling developed in this thesis by simulating polymorphic audio and video application threads.

1.4 Thesis Contributions

Future embedded multicore architectures will have compute resources with diverse energy-delay profiles. Current design methodologies in SoC operating systems and development tools limit the exploitation of diverse compute resources for energy and delay optimization. Our goal in polymorphic computing is to provide a large spectrum of energy-delay choices using few design points in the energy-delay space. We achieve this by proposing polymorphic threads which can execute on more than one style of compute resource by morphism evolution. We propose user satisfaction model for the application layer and network layer as an optimization goal. Scheduling of threads and allocation of resources based on user satisfaction is our solution to exploit the diverse energy-delay compute resources offered by embedded multicore architectures. The main contributions of this thesis are :

- We have developed two polymorphic applications namely, audio and video decoder, for evaluating the ideas developed in this work. The multi-threaded applications are implemented in C++. Application threads are parametrized to have two morphisms - CPU and FPGA.

- We have developed a heterogeneous multicore simulation framework, PolyNoC, for application layer user satisfaction optimization. User satisfaction model based on sigmoid function is built into the scheduling layer. Application threads use PolyNoC APIs to interface with our simulation framework.

- The user satisfaction model proposed for application layer is extended to the NoC routers for developing virtual channel allocation heuristic. We develop an energy model to evaluate the effect of resource allocation on user satisfaction under scenarios where energy is at a premium.

- Our dynamic binary translator, TCM framework, is also integrated into the PolyNoC simulation engine to enable designers to experiment with dynamic optimizations using *wrappers* embedded in applications.

1.5 Organization

The remainder of this thesis is organized as follows. The reader can find brief review of prior research literature related to this work in Chapter 2. We discuss energy-delay space expansion in the context of polymorphic system design in Chapter 3. User satisfaction model for application layer and network layer is developed in Chapter 4. Chapter 5 provides a detailed description of PolyNoC simulation framework. User satisfaction model for routers in the network layer is discussed in Chapter 6. In Chapter 7, the integration of TCM framework with PolyNoC is described and evaluated. The thesis is concluded in Chapter 8 by noting our contributions and suggesting possible future extensions.

CHAPTER 2. REVIEW OF PRIOR RESEARCH LITERATURE

A brief background of multicore architectures is presented in Section 2.1. Section 2.2 reviews prior work on pertinent issues addressed by past research related to network on chip communication paradigm. The effect of modeling user satisfaction on performance optimization in embedded systems is reviewed in Section 2.3. Operating system design issues addressed in previous research is reviewed in Section 2.4.

2.1 Background

Monotonic trend in microprocessor design metrics over the past few decades is largely attributed to transistor scaling. Steady increase in clock speed, instruction throughput and transistor count can be traced to advances in fabrication technology. Faster transistors, facilitated by scaling of transistor gate lengths, have resulted in 300x increase in clock frequency as evidenced from the data for Intel x86 processors over the last three decades (Figure 2.1).

Similarly, transistor count has doubled every two years in accordance with Moore's law [Moo98] due to the result of aggressive fabrication techniques as illustrated in Figure 2.2. The positive trend of Figure 2.1, due to scaling, has been counteracted by lack of proportional scaling in electrical parameters of the transistor such as threshold voltage, leakage current, interconnect parasitic and supply voltage. The lack of reduction in interconnect parasitic and leakage current is particularly problematic when scaling approaches 22nm technology node and beyond.

Power dissipation and power density have not scaled in step with increase in transistor density shown in Figure 2.2. Power depends on supply voltage which has not seen the same scaling factor used to reduce transistor gate lengths. Cooling demand for high performance

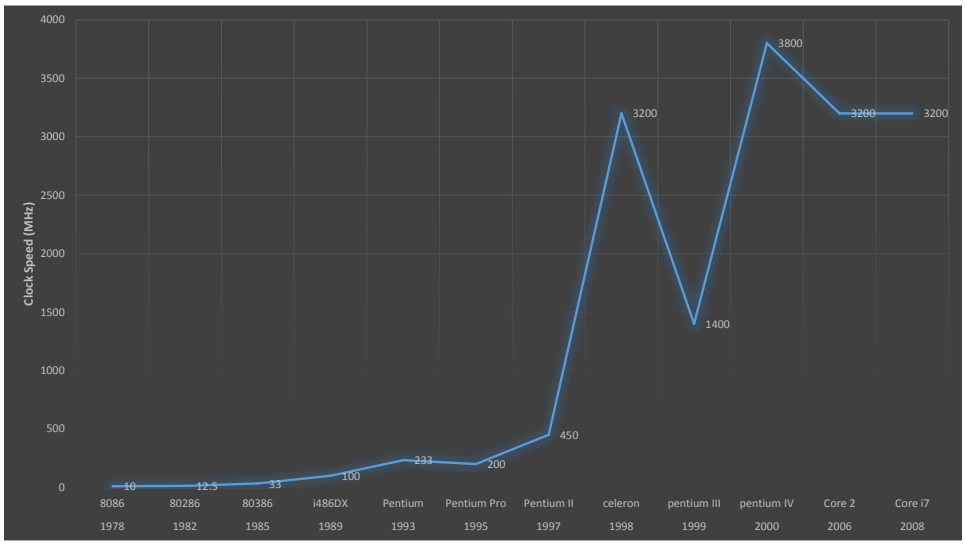

Figure 2.1: Clock frequency trend for Intel x86 Microprocessor Series

servers and shrinking power budget of mobile chips are some reasons which forecast lower power requirement specification in the future. For a fixed power budget, the doubling of transistors at every technology node ensures that half the transistors will be *dark*.

Designers have adopted several techniques to combat power wastage. Clock gating, sleep states, dynamic voltage/frequency scaling are representative of digital design methods which attempt to minimize dynamic power due to unnecessary transistor switching activity. Discovering *redundancy* in implementation and harvesting the power savings subject to minimal overhead is the underlying strategy. It is imperative to exploit redundancy at all levels of system design - applications, operating system, architecture and digital design to satisfy stringent design constraints. Research addressing digital design such as logic synthesis, floor planning, timing analysis have been automated with CAD tools and these tools produce well optimized designs for performance, power or some combination of these. Continued innovations in computer architecture aimed at circumventing *power wall* and *stagnation of clock frequency* have ushered in homogeneous and heterogeneous multicore chips as potential solutions. Homogeneous and heterogeneous chips have seen wide spread integration in both high performance and mobile computing systems. Multicore architectures can be broadly classified based on the type

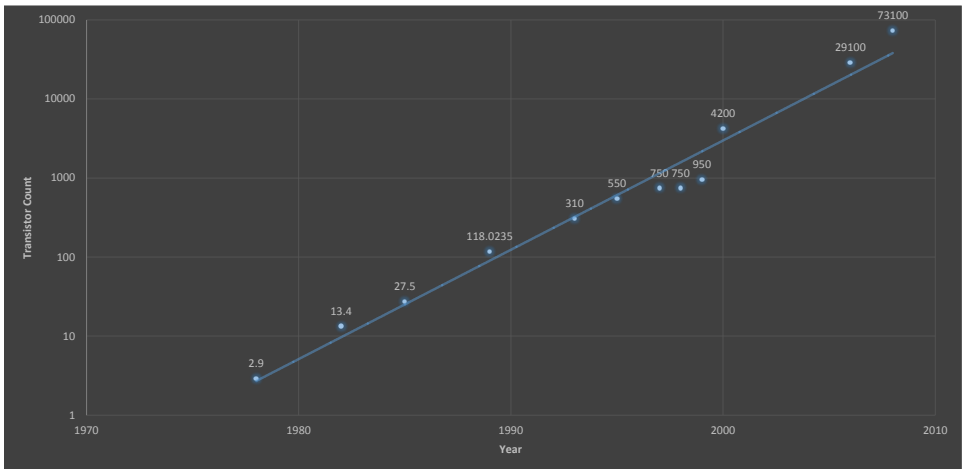

Figure 2.2: Transistor count trend

of compute cores and the connectivity between cores. Homogeneous multicore architectures use the same type of compute core for all the nodes. Alternatively, heterogeneous multicore architectures use variegated compute cores with multiple energy-delay profiles. Connectivity supported between cores can be crossbar, ring, taurus or mesh. For example, Graphics Processing Unit (GPU) architecture [ND10] employ array of identical stream processors as the compute cores. A hardware scheduler manages mapping threads to stream processors. CUDA framework simplifies programming of GPUs which has led to the widespread acceptance of this homogeneous multicore architecture. The OMAP 4 [TI] embedded processor series from Texas Instruments uses various compute cores optimized for different applications. Two ARM cores, a graphic accelerator and an image signal processing engine integrated in TI OMAP 4 are spread out in the energy-delay spectrum.

Multiple compute cores offer larger design space for optimizing application performance and power consumption. Compute cores differing in energy-delay characteristics further widens the design space. Thus, all levels of system design need to explore favorable optimization points in the enlarged design space. Applications must be aware of multiple compute cores. Multi-threading is a technique in which independent threads can execute simultaneously by identifying redundancy in data. Threads can be *morphed* to execute on various compute cores such as FPGA, GPU or CPU. Mapping application threads to this design space is a challenge feverishly addressed in research. Operating system is yet another component of the ecosystem

which need to be modified to be aware of the multicore model. Traditional techniques of designing each layer separately also impacts the overall design goal where each layer may have conflicting optimization goals and costs. A uniform cost metric at multiple design abstraction layer will benefit the system as a whole.

Mobile systems are optimized for a high degree of interaction from the end users. End user essentially becomes part of the mobile ecosystem. Further, humans are a major source of redundancy which could be exploited for optimizing performance and energy. The argument for customized architecture [SAW+10] is strong in mobile systems when subject to the constraints of balancing power and performance for varying user behavior.

Research efforts related to mobile computing systems touches on issues in multicore architecture exploration for performance/power, thread scheduling/mapping heuristics at operating system layer and modeling user interaction. The following sections will provide an overview of related research work.

2.2 Issues in Network on chip design

NoC based system on chip (SoC) design has been studied extensively in research literature. The authors of [DT01] advocate using NoC communication paradigm to overcome low global wire utilization and other issues. Efforts to estimate the overhead of network logic required for *packet* based information exchange between NoC *nodes* is presented in [DT01]. The presence of shared network resources increases the utilization factor. Dedicated global wires between top level system modules/nodes suffer from poor utilization during idle times. Networked global wires on the other hand are shared among all the nodes of the NoC and hence, the utilization factor is increased. They [DT01] report an area overhead of 6.6% for the added network logic per NoC node. Buffer resources, arbitration logic, virtual channel allocation policy, routing strategy, flow control logic and crossbar allocation are major functional elements of the *router* interface. Virtual channel (VC) abstraction mitigates the effect of locking physical buffering resources by packets from a single communication flow. Downstream events such as unavailability of input buffer credit and congestion can block egress of packets from the current router. Hence, network throughput is heavily dependent on packet egress rate of routers in

the flow path. Packet egress rate depends on VC allocation policy to a large extent and subsequently, a large fraction of research is devoted to study of effect of various VC allocation policies on network throughput. The use of VC in [Dal90] increases the throughput of network by a factor of 4 and also reduces the dependency of throughput on network depth. A detailed theoretical analysis of VC based flow control and experimental evaluation of latency, throughput for various traffic conditions is also given in [Dal90]. Architecture of router determines latency and throughput of packets transiting through to destination NoC node. Pipelined VC router logic improves throughput by 40% over wormhole router in the data presented in [PD01b, PD01a]. A router delay model which parametrizes flow control method, cross bar delay due to sharing, virtual channels is also presented in [PD01b, PD01a] to validate claims of efficiency of VC based pipelining of router logic. Experiments using the router delay model of [PD01b] predicts per NoC node latency is constant upto 8 VC per physical channel and hence, the result supports using VC for better throughput at minimal increase in per node latency. The literature work in [DT01, Dal90, PD01b, PD01a] should provide a wholesome overview of NoC router function and design specifications. Several literature surveys [TLHC12, BM06] will enlighten the reader to better understand NoC based design issues.

Mapping computation to NoC nodes is an optimization problem at the operating system layer. Application threads will be assigned to NoC nodes depending on the compute resource in need (such as FPGA, CPU, GPU), congestion conditions, hops needed to reach destination. Energy per packet traversal is directly proportional to path length in the NoC. Mapping algorithms in prior research optimize number of hops between communicating threads to minimize energy for communication. It is beneficial to map threads to NoC nodes which are not in congestion zone even if traversal path length is longer. Longer traversal path length will result is higher initial latency but higher throughput if there is minimal congestion. However, packets may spend longer time in buffers waiting for egress ports in congested paths thus increasing latency while decreasing throughput at higher energy expense per packet. Thus, tradeoffs exist between number of hops, energy per packet, latency and throughput at the mapping phase. The authors of [HM04a] propose an energy aware scheduling algorithm for statically mapping and scheduling an application's tasks on heterogeneous NoC. They formulate an optimization

problem dependent on energy of computation per NoC node and energy of the volume of bits communicated between two tasks. A mapping of tasks to NoC node which results in minimization of the total energy over all computations and communications is the optimal solution. The authors develop heuristics to arrive at sub-optimal solutions due to NP-hardness of the original optimization problem. Experiments in [HM04a] on multimedia applications demonstrate an average energy savings of 44% when compared to task mappings generated by earliest deadline first scheduling strategy.

2.3 Issues in User satisfaction modeling

User satisfaction has been difficult to quantify due to its dependence on human perception unlike other design parameters like delay, power, energy. Despite this drawback, user satisfaction remains an important design parameter for highly interactive embedded systems such as smart phones and hand held devices. The authors of [SOM+08] develop a model that relates user satisfaction with measurements of performance counter values for a representative class of applications. They build a neural network model from observations of performance counter to predict user satisfaction at run time. Predicted user satisfaction is further used to vary the frequency and voltage of the processor to reduce CPU power consumption. A 25% reduction in CPU power is reported for a class of applications using this methodology. Though the authors of [SOM+08] quantify user satisfaction in terms of architectural parameters, they do not discuss techniques to improve user satisfaction or suggest novel architectural solutions. In [CM10a], an user centric design methodology is proposed for NOC type architectures. The authors of [CM10a] collect behaviour traits of users for various applications and cluster them into different classes. They claim to optimize system design to better adapt to user requirements for each class. The goal is to balance workload variations according to user experiences in embedded systems. An average energy savings of 30% is reported in [CM10a] using this technique. The authors of [CM10b] propose a dynamic scheme for allocating applications tasks to embedded multiprocessor considering the end user behaviour while making resource allocation decisions. They claim to have achieved 70% communication energy savings by minimizing the communication cost in NOC implementation of multiprocessor embedded system. The work

in [ZPT14] also uses user satisfaction modeling to answer questions about the effect of limited energy source on scheduling behavior in polymorphic framework.

2.4 Operating system design for Heterogeneous computing

Heterogeneous computing systems present several challenges for designing operating system components. Hardware platforms in which reconfigurable fabric is interspersed with high performance CPU cores require thread abstractions for both compute styles. The H-threads programming model for FPGA-CPU hybrid computing hardware is described in [ANA04] and [PAA+06]. They have introduced the abstraction at process level. The H-threads programming model abstracts FPGA/CPU components to form a unified multiprocessor architecture platform. Hence, the designer is freed from specifying and embedding platform specific instructions for communicating across the Hardware/Software interface. However, the H-threads model falls short of investigating the important parts of embedded system design such as the scheduler, dynamic binding of unit of computation, virtualization and relocatability. The polymorphic design paradigm described in this thesis addresses the effect of choosing computing styles on the application performance at run time. Another limitation of H-threads is that it is designed for a single bus based system unlike polymorphic paradigm where we consider more powerful NoC based embedded systems. The other relevant works on operating system design for heterogeneous platforms are discussed in [MNC+03], [NAE+08], [NCV+03]. The work in [MNC+03] explores the relocation issues between hardware and software tasks by implementing a relocatable video decoder. They find that reconfiguration delay in FPGA is an impediment to performance during hardware task relocation. The decoding rate achieved falls in the range of normal video playback providing proof that even with reconfiguration overhead it is possible to effectively use heterogeneous computing systems under polymorphic abstraction. CUDA programming model [Cud] uses compiler directives to marshal parallel sections of the application to the GPU computing cores in Tegra [Nvi] like heterogeneous computing platforms. The hardware scheduler takes care of allocating resources to groups of threads which follow the same path in the control flow graph. The design of GPU scheduler is not open source.

Mobile OS frameworks such as Android modify existing Linux scheduler to balance resource allocation between interactive and background threads. CPU time slice is the primary resource managed by the Linux scheduler and it is allocated to threads based on a combination of heuristics, priority and notion of time. Linux kernel 2.6 ushered in the O(1) scheduler to provide an upper bound on the time required to pick the next thread which will get the CPU time slice in the current scheduling epoch. An active queue holds threads with unfinished time quanta and an expired queue holds threads with completed time quanta in the current scheduling epoch. The queues are swapped once all threads in active queue move to expired queue after expiring of time quanta. Swapping starts a new scheduling epoch. However, interactive threads face the danger of unpredictable delays by waiting in the expired queue until all threads in active queue migrate to the expired queue. Further, the decision to place threads in priority queue used heuristics such as ratio of sleep time to CPU time to gauge the interactivity level of the thread, processor affinity etc. Linux 2.6.21 introduced the *Completely Fair Scheduler* (CFS) [CFS] to address the unbounded latency problem of threads in the expired queue and also being fair to all threads without using too many heuristics. CFS assigns an unfairness value to each thread which is in need of CPU time. Unfairness measures the amount of time a thread spends waiting for CPU time. CFS orders threads based on unfairness using a red-black tree data structure. The thread treated most unfairly is on the leftmost node of the tree and is picked by CFS in O(log N) time to execute next by following left most path for N ready to run threads. Inserting new threads and adding waking up threads are also O(log N) time in the red-black tree structure. This scheme mimics a multitasking CPU executing N concurrent threads each getting only $\frac{1}{N}$th of the CPU power. The notion of fairness is simulated using the unfairness value for waiting threads which get priority for execution in future time slices of the current scheduling phase.

CHAPTER 3. POLYMORPHIC COMPUTING SYSTEM

Mobile platforms have become omnipresent. They serve as terminals to access services from the cloud. Compute intensive applications such as searching have been migrated to high performance systems in the cloud backbone. Interactive applications on mobile devices access various services such as searching, streaming, storage etc. from the cloud. Interactive applications impose tight design constraints [App14] on hardware architectures for embedded systems. This makes the traditional performance (time or throughput) driven resource allocation outdated for these devices.

Within a mobile system context, energy efficiency is of paramount importance. The delay or response time is also critical in the mobile devices. Fast Launch, Short Use Apps is how Apple [App14] characterizes a typical iPhone/iPad app in its iOS Application Programming Guide. In fact, UI responsiveness is one of the holy grails of app development emphasized time and again in all Apple and Google Android literature. If an app does not respond within 5s, iOS will terminate it, and Android will issue an ANR (App not responsive) notification. This is how Apple iOS programming guide describes an app: The strength of iOS-based devices is their immediacy. A typical user pulls a device out of a pocket or bag and uses it for a few seconds, or maybe a few minutes, before putting it away again. ... Your own applications need to reflect that sense of immediacy by launching and getting up and running quickly. If your application takes a long time to launch, the user may be less inclined to use it..

Embedded systems have to be adaptable and scalable to meet the unique resource demands of such applications to deliver satisfactory performance. Effective sharing of system resources by content consumption applications (streaming, storage) is imperative for user satisfaction. For example, to keep energy requirements of video applications within the energy budget, several compute intensive functions such as discrete cosine transform (DCT), motion estimation are

implemented in custom hardware in embedded chips. When two or more applications compete for the hardware DCT resource, the availability of software version of DCT will enable better sharing of the hardware DCT with graceful performance degradation for the applications which do not get to use hardware DCT. Such tradeoffs can be generalized by allowing computation to execute on more than one style of compute resource and this paradigm of computing is referred to as *Polymorphism* in this thesis.

The importance of energy efficiency in a portable personal device struggling to serve the user interaction needs for a full day on a single battery charge cannot be overstated. Energy, however, cannot be the sole focus of polymorphic embedded system. The application performance and response time do play a significant role in user satisfaction as evidenced by limits on response time of application threads in existing mobile operating systems. In order to balance the energy-time needs for multiple applications, a mobile platform needs more flexible energy-time design spaces for the applications. This work argues that polymorphic systems offer such design space flexibility in a relatively straightforward manner.

3.1 Polymorphism

Polymorphic computing paradigm emerges as a direct consequence of heterogeneous computing architecture with programmable cores. In such computing platform, an *unit of computation, UoC* (such as process, thread, function) need not be statically bound to either a hardware or software implementation. For example, consider an application A_1 designed with 5 threads $(T_1, T_2, \ldots, T_5)$ and application A_2 with 4 functions $(F_1, F_2, \ldots, F_4)$. In this example, the unit of computation for application A_1 is *threads* and for A_2 it is *functions*. In this case, we argue that thread T_1 of A_1 and function F_1 of A_2 have no reason to be statically implemented as software or hardware. This is the traditional desktop paradigm for software applications that is being currently used for embedded systems. The crux of our paradigm is such bindings can be made *dynamically*. We believe that dynamic binding of unit of computation to a core of NoC platform will efficiently utilize resources and it takes application design to the next level as compared to traditional desktop application design methodology.

With heterogeneous multicore nodes, an interesting notion of polymorphic threads arises. A thread T_i can be compiled or programmed for binary images applicable to multiple core types - CPU, GPGPU, RC (reconfigurable core - FPGA). We call such a thread a *polymorphic thread* with $T_{i,j}$ representing jth morphism for a polymorphic thread. A polymorphic application, A, can be realized as a thread flow graph, directed acyclic graph (DAG), of polymorphic threads $T_0, T_1, \ldots, T_n$. An edge (T_i, T_j) shows dependence with T_i as producer and T_j as consumer. Each thread T_i may be programmed with $l_i + 1$ morphisms $T_{i,0}, T_{i,1}, \ldots, T_{i,l_i}$. Each thread morphism $T_{i,j}$ has radically different energy-time $E * T$ profile $E\,T_{T_{i,j}}$.

Consider the Android framework as an example to illustrate the application of polymorphic threads. In Android framework [And], an application is designed as a collection of activities. Each activity is a slice of model-view-controller (MVC). An activity can be instantiated by other apps and system components. We can view an Android app as an activity flow graph along the lines of the thread flow graph, even though each activity may not have its own thread. Activities are typically invoked through the Android messaging framework called *intents*. The intent layer within the Android framework gets to intervene preceding each activity invocation in much the same way as the OS scheduler does for a thread. Hence, even if each activity is not encapsulated as a thread, the morphism selection could still be performed dynamically within the intent layer of Android framework.

To support dynamic bindings and communication among the units of computation, the NoC system has to be *relocatable* [KPT14]. Relocatability virtualizes the NoC platform in the same way physical memory is virtualized in desktop applications. We have developed such an NoC simulator to assess the performance and placement metrics of video and audio benchmark applications. The design of *polymorphic thread scheduler* is explored for a NoC system and evaluated in this thesis. The importance of a scheduling policies becomes apparent when we think of the several morphisms for the unit of computations that can potentially exist in the system.

3.2 Polymorphic Framework

Polymorphism is defined in energy, time, user satisfaction metric space. A polymorphic system has two components.

1. **Polymorphic Architecture** – This is often realized as heterogeneous cores in a multi-core system-on-chip (SoC). A given program specified behavior is realizable along different energy-time points with different architectural or micro-architectural style cores. For the same program, energy-time characteristics on a 4-wide superscalar core are different than on a GPGPU core or on a reconfigurable FPGA-style core. In a traditional single processor system, such energy-time polymorphism can also be realized to a limited extent through dynamic voltage frequency scaling (DVFS).

2. **Polymorphic Applications** – A polymorphic SoC does no good if the programmer does not compile some of the threads of an application into multiple morphisms suitable for the morphisms expressed by the architecture. A program or a thread that is compiled into multiple morphisms is called polymorphic. A polymorphic thread need not necessarily be driven by architectural polymorphism. The classical algorithmic polymorphism - quick sort versus bubble sort can also create different morphisms along the energy-time design space. The OS scheduler not only has to schedule a thread, it has to pick a morphism of the thread to be scheduled in order to achieve certain energy-time characteristics for the system.

3.3 Design space

Consider a program/thread designed for two morphisms, $M_1 = (E_1, T_1)$ and $M_2 = (E_2, T_2)$. For instance, the morphism M_1 could correspond to Discrete Fourier Transform (DFT) on a CPU core and M_2 could correspond to DFT on a reconfigurable core. In reality, each of these morphisms represents an energy-time curve parametrized by some algorithmic parameter, and not a single point in the energy-time space. For DFT of n samples, (E_1, T_1) is likely to be $(c_{E_1} * n^2, c_{T_1} * n^2)$ for architecture and algorithm dependent constants c_{E_1} and c_{T_1}. The point

(E_2, T_2) maps into similar n^2 curves with different constants. If there was algorithmic morphism say Fast Fourier Transform, the (E, T) point will map to curves representing $n * log(n)$ with appropriate constants.

With polymorphic scheduling, even though a programmers efforts may be limited to only two morphisms M_1 and M_2, the operating system polymorphic scheduler can achieve any design point represented as a linear combination of M_1 and M_2 – $\alpha_1 * M_1 + (1 - \alpha_1) * M_2$, the line connecting points M_1 and M_2 as illustrated in Figure 3.1.

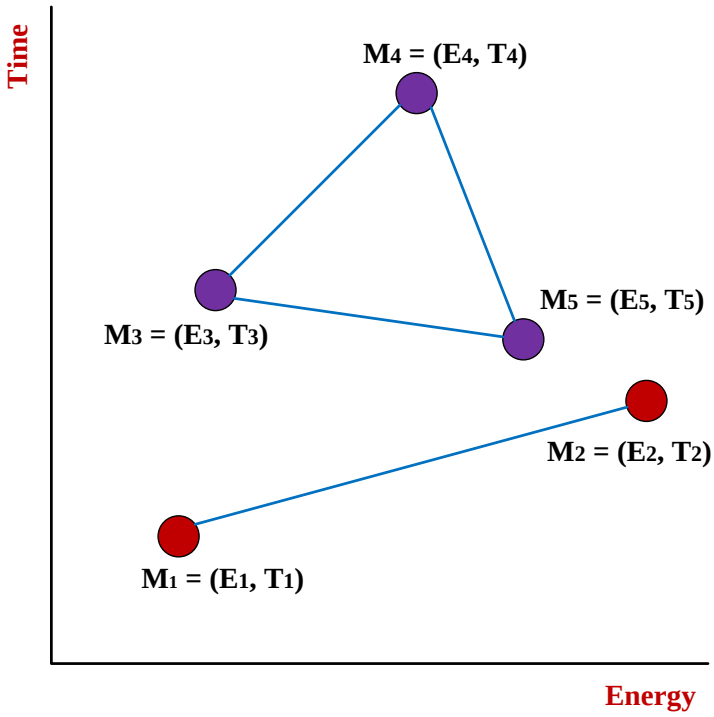

Figure 3.1: Linear combinations of morphisms

The parameter α_1 represents the relative scheduling frequency for morphism M_1. This is a significant amplification of the two point design space provided by the programmer through polymorphic framework. Similarly, a three morphism point programmer energy-time design space is amplified into a triangle (also shown in Figure 3.1). In general, an N point design space is mapped into its convex hull. Such expanded energy-time design spaces give the mobile systems more flexibility in meeting system wide energy, time, and user satisfaction goals.

3.4 Energy-Time Design space for video decoder

Video decoding is computationally intensive. Kernels of algorithms in the decoding engine is used in several multimedia applications. The decoding phases can be demarcated into distinct sub-tasks with well defined interface. The sub-tasks have wide range of implementation choices in software and hardware [VZT03]. Hence, video decoding application is an ideal candidate for polymorphic implementation. The sub-tasks of video decoder pipeline are shown in Figure 5.8. A hardware/software co-design of MPEG-2 decoder evaluated in [VZT03] is used to illustrate the energy-time design space expansion of polymorphic paradigm. Table 3.1 and 3.2 lists the morphisms for DCT and Q threads in the MPEG-2 decoder application.

Table 3.1: MPEG-2 Decoder IDCT Morphisms

Thread	Morphism	**Energy(mJ)**	**Time(ms)**
DCT	SW	32.34	15.61
	HW1(1-dct)	0.156	0.0075
	HW2(2-dct)	0.709	0.015

Table 3.2: MPEG-2 Decoder Quantization Morphisms

Thread	Morphism	**Energy(mJ)**	**Time(ms)**
Q	SW	10.17	13.45
	HW	14.09	3.2

A software morphism (SW) of DCT (CPU resource) consumes 32.34 mJ of energy and completes the task in 15.61 ms in Table 3.1. DCT can also be implemented in FPGA hardware as one dimensional DCT (1-DCT, HW1 morphism) with energy consumption of 0.156 mJ and delay of 0.0075 ms. Another morphism of DCT (2-DCT, HW2) in Table 3.1 consumes 0.709 mJ of energy and has a delay of 0.015 ms in FPGA hardware. Similary, the quantization (Q) phase of the MPEG-2 decoder has two morphisms - one in software (SW) and the second one in hardware (HW). The energy consumed and delay in each morphism is listed in Table 3.2.

The OS scheduler can choose morphisms for the Q thread depending on available resources. The ET space for Q thread depends on the fraction of time each morphism is used during the life time of the MPEG-2 decoder application. Figure 3.2 plots the energy consumed and delay

of the decoder application for various fractions of SW and HW morphisms. The upper surface plot is the energy space and the lower surface plot is the delay space.

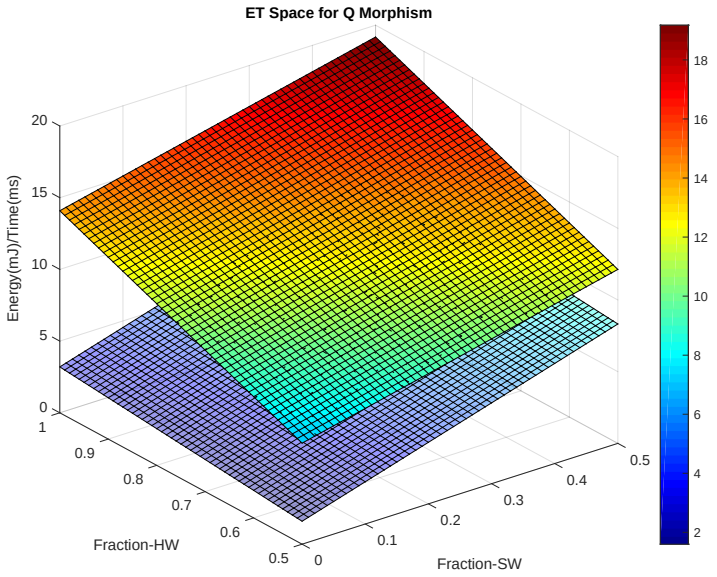

Figure 3.2: ET Space variation for Q thread morphisms

The energy space and delay space for DCT morphisms is plotted in Figure 3.3. The plots show the variation in energy and delay at a specific fraction for each morphism. The graph is a 3D graph sliced at a given fraction along each axes to show the variation of energy and delay along the slice.

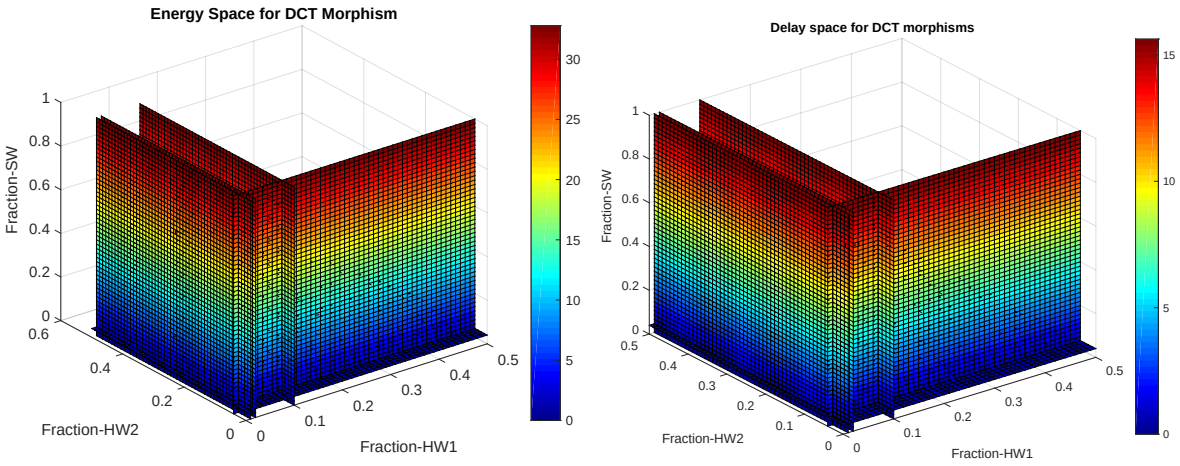

Figure 3.3: ET Space variation for DCT thread morphisms

Heterogeneous SoC and Polymorphic paradigm can potentially expand the ET space of polymorphic applications as evidenced by the MPEG-2 decoder example. The next section details architectural design of existing Heterogeneous SoC which can be modeled using the polymorphic network on chip simulator developed in this work.

3.5 Heterogeneous SoC

Mobile chipsets are gravitating towards multicore SoC architectures and each successive generation has more compute cores. For broader energy-time space support, these cores need to be heterogeneous. We believe that such heterogeneity must include not only the traditional Von-Neumann cores, but should also include non-traditional FPGA fabric based reconfigurable cores (RCs), custom hardware (ASIC like units), and general purpose graphics processing unit cores (GPUC). The reconfigurable cores and GPGPU cores are lighter-weight, smaller granularity versions of the FPGA and GPGPU respectively. An interconnection network facilitates communication between these cores.

Xilinx [Xil] has recently introduced Zynq (Figure 3.4) which contains elements of such an architecture. It contains an ARM Cortex A-9 dual-core processor along with the traditional FPGA cells such as LUTs, DSP cores and block RAMs. FPGA logic slices are versatile for implementing wide range of hardware which can exploit bit parallel execution. The Zynq SoC interfaces the programmable logic with the ARM subsystem through the AMBA bus. Applications which can benefit from hardware/software co-design can benefit from Zynq SoC.

iVeia [iVe] has released a prototype tablet system based on Zynq chip. The polymorphic simulation platform developed as a part of this thesis is an tool to simulate designs on iVeia prototype for initial design exploration.

The Tegra 4 SoC [Nvi] from Nvidia is yet another heterogeneous architecture used widely in Android based phones and tablets. Tegra 4 has four ARM Cortex A-15 CPUs and one battery saver ARM Cortex core for general purpose computing. General graphics applications such as 3D games can be accelerated by the 72 GPU cores. Further, custom hardware such as video decoder/encoder and audio can be used by multimedia applications. The battery saver core works to save energy during device idle times by running background applications such as

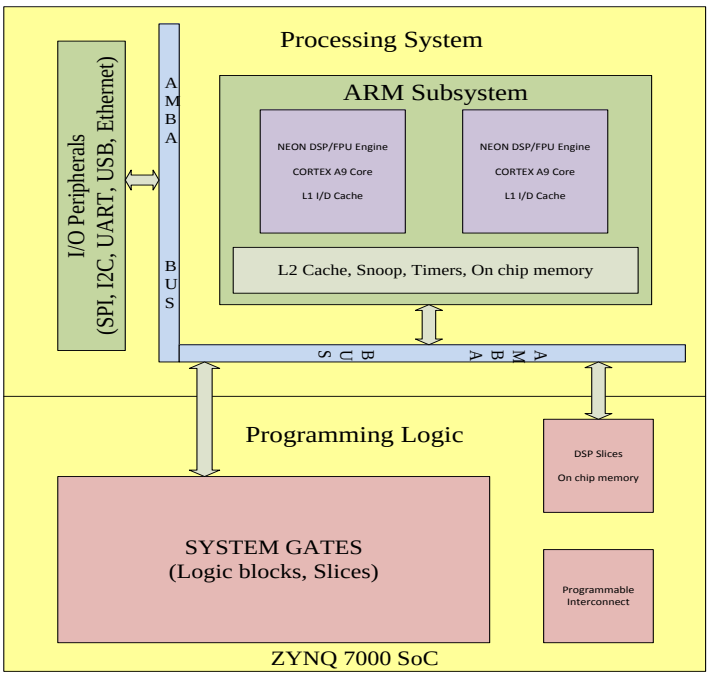

Figure 3.4: Xilinx Zynq 7000 SoC

sending/receiving e-mails. The Tegra 4 SoC is illustrated in Figure 3.5.

Glimpses of polymorphic paradigm can be observed in the architectural decisions in Zynq and Tegra SoC. The idea is application threads can use more than one compute resource to expand the energy-performance design space. Polymorphic NoC simulation model developed in this work can be used for modeling and exploring heterogeneous SoC architectures discussed in this section.

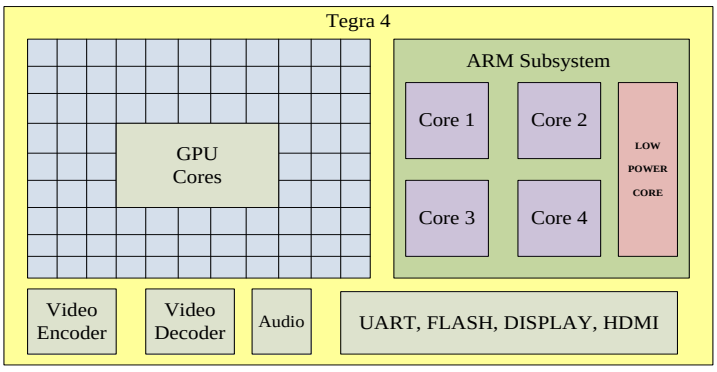

Figure 3.5: Nvidia Tegra 4 SoC

CHAPTER 4. USER SATISFACTION MODEL

The focus of this chapter is to introduce user satisfaction as a primary metric for resource allocation in a mobile operating system such as Android. Mobile OS uses touch and multimedia inputs that mimic real world actions such as swiping, tapping, pinching, and speech to manipulate on-screen objects. They are user centric as their sole focus is on user interaction through a graphical user interface. Traditional average case performance (delay or throughput) driven resource allocation will be restrictive for these mobile devices. A user satisfaction metric which models the user-centricness of these devices will be better suited for resource allocation. We develop a user centric scheduling layer incorporating energy-time user satisfaction trade-offs within the polymorphic simulation framework. Polymorphic thread scheduling in our framework is driven by optimizing user satisfaction. User satisfaction of application is subjective and often varies among individuals. However, simple models of user satisfaction for restricted class of applications such as multimedia can be evaluated using scheduling policy and compared with traditional earliest deadline first, first come first serve policies. We experimentally show that user satisfaction scheduling policy performs better when compared to traditional scheduling policies on NOC style diverse resource architectures.

Humans seem to have saturating interactions with our environment. Up to a certain point, increased quality of interaction - perhaps through increased allocation of resources, leads to improvement in user satisfaction. However, at some point, our satisfaction saturates. Any further enhancement of resources to enhance the user experience at this point leads to negligible gains in real user satisfaction. The same holds for dis-satisfaction or user satisfaction at the negative end. Once a user is dis-satisfied (saturated), making the resource allocation worse has no real impact on the user satisfaction. A second property of user satisfaction is that in the

unsaturated region, it grows non-linearly. We model user's perceived satisfaction from resource allocation with a function called *user satisfaction*.

4.1 Modeling User Satisfaction

Human perception is important since it is eventually the perceived satisfaction from the application that matters. Due to the natural signal processing limits of HAVS, often there is a *lower knee* on the performance below which no satisfaction is derived. This establishes a lower bound on the performance metric, below which, there is zero or no user satisfaction. There is an *upper knee* after which the marginal satisfaction of any additional performance is minimal. This establishes an upper bound on the performance metric, above which, there is minimal improvement in user satisfaction for more resource allocation. It is the middle region between these two knees that is important with nonlinear progression in satisfaction with increasing performance. Voice perception clamped within the perceptible frequency range has this characteristic. User satisfaction that fits such profile is best captured with a *sigmoid function*.

The sigmoid model for user satisfaction (UserSat) is mathematically expressed by Equation 4.1.

$$UserSat(t) = \frac{1}{1 + \alpha\ e^{-\beta t}} \tag{4.1}$$

The parameters α and β are used to define the characteristics of the sigmoid curve in Figure 4.1. Parameter β is referred to as the *slope* of the linear region and controls the sensitivity of user satisfaction to the change in resource allocation. The saturation regions at the upper and lower knee of the sigmoid are controlled by the parameter α. The plot in Figure 4.1 has two sigmoid curves for different values of the parameters α and β to illustrate the effect on the sigmoid shape. The variable t is the physical parameter of interest such as throughput at the application layer, bandwidth of flits at the network layer etc. which influence the user satisfaction. It is constrained by the resource allocation policies.

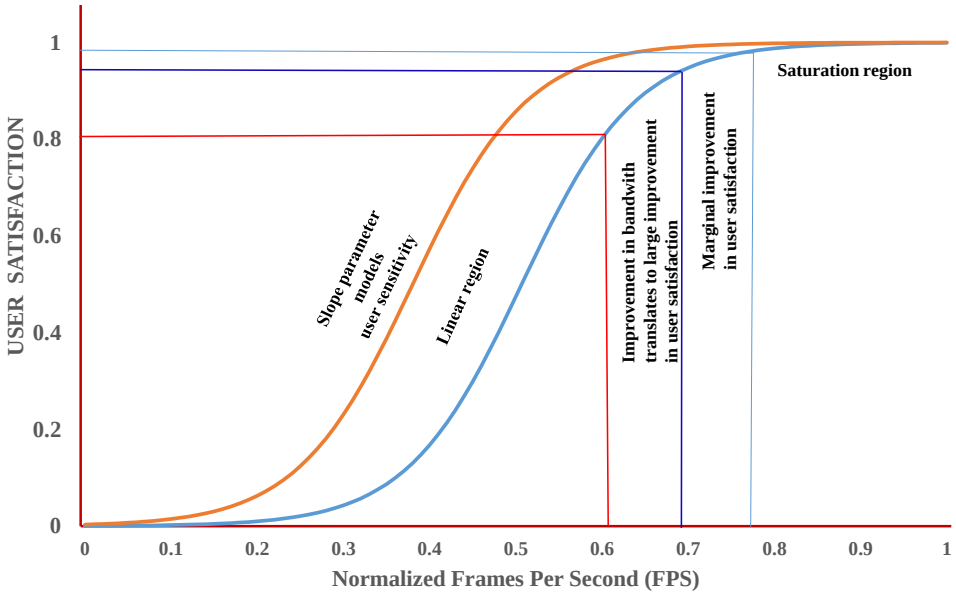

Figure 4.1: Sigmoid Function for User Satisfaction Function

4.2 Application Layer User Satisfaction

The following examples illustrate modeling of user satisfaction using sigmoid function. The sigmoid function is the classical S shaped curve expressed as $u(t) = \frac{1}{1+e^{-t}}$ for $\alpha = 1$ and $\beta = 1$. The parameters of an application can be continuous or discrete. Frame rate in video playback is a discrete parameter since digital video sampling is done discretely unlike the web page loading delay of a browser application, which is a continuous parameter. Frame rate is one of the parameters that affects user experience to a large extent in any form of visual multimedia applications. Such applications can include movie players, streaming video, teleconferencing, video chatting, gaming engines etc. The user will specify the satisfaction expected from an application in the range of [0,1] for parameters of interest in the application and in this case it is the frame rate of the video multimedia being viewed by the user. Frame rate range will be specified by the application designer since each type of video application can operate in a different range. Video player which plays movie files/DVD can have frame rate in the range of 15 frames per second(fps) to 30 fps in steps of 2 fps as supported by the hardware platform. 3D Gaming applications need 30 fps to 60 fps in steps of 5 fps and video chatting has frame rate of 3 fps to 15 fps in steps of 1 fps. Let the frame rate of 3D gaming be expressed by

Equation 4.1 like model. Then its user satisfaction can be modeled by the sigmoid function $u(S) = \frac{1}{1+e^{-\frac{(S-30)}{5}}}$ for video playback applications

This is system dependent and has to be specified by the designer. Sigmoid modeling is ideal for video player since below 15 fps the user satisfaction is close to 0 and above 30 fps it is close to 1 and in between there is close to linear increase in user experience. The application interface consists of a GUI which the user can use to specify the satisfaction function by drawing the sigmoid and the back end logic can interpolate the sigmoid function to derive the user satisfaction values for each of the acceptable frame rates in the application. Though the users sigmoid function looks continuous, interpolation will make the sigmoid function look piece-wise continuous.

4.3 Polymorphic Thread Scheduling

Polymorphism introduces new challenges for scheduling threads in a mobile platform with one or more multi-threaded applications. Applications vary in nature and resource requirements. Video decoder output is measured using frame rate and resolution while an audio decoder output is measured in samples per second. Traditional goals such as performance are optimized by scheduling threads as early as possible when resource constraints are satisfied. Such scheduling policies ignore user's perception of satisfaction from resources allocated to application threads. A relative mapping of satisfaction of the user to various outputs of the application in the range (0,1) is modeled by user satisfaction. Intuitively, the dynamic distribution of resources to application at different outputs is guided by user and used as a priori information to the scheduler. Further, user satisfaction is a common denominator for comparing resource allocations to application outputs which cannot be otherwise balanced with the same scale. Frame rate for video and samples per second for audio are like apples and oranges. However, from a user perspective, the HAVS interacts with these applications and determines the satisfaction level from the application outputs at any given time period.

4.3.1 Thread Communication Graph

An application is partitioned into one or more threads which may have dependencies, communication bandwidth and resource requirements. A set A of n applications, $A = \{A_i | 0 \leq i < n\}$, is used for the following discussion. A_i is associated with a thread communication graph (TCG) which determines the dependencies, communication degree and resource needs of threads. The morphism choices for each thread is also specified by the designer in the TCG. Each thread created will be associated with a structure, indexed by the thread identifier, that contains information about morphism, resource needs (CPU, GPU etc.) , communication, execution state (Run, Wait, Ready) among other details. Application A_i consists of p_i threads $T_{i,j}$ for $0 < j < p_i$. The morphism space for thread $T_{i,j}$, $0 \leq j < p_i$, is denoted by $T_{i,j,r}$ where $0 \leq r < m_{i,j}$. Here, $m_{i,j}$ denotes the morphism space for the thread $T_{i,j}$. Figure 4.2 shows the thread communication graph for an application with 7 threads, $T = \{T_{0,1}, T_{0,2}, T_{0,3}, T_{0,4}, T_{0,5}, T_{0,6}, T_{0,7}\}$. Any of these threads can be implemented as a software thread or as a hardware thread.

The graph in Figure 4.2 clearly captures the computational dependencies and communication flow between threads in an application. Nodes in the graph represent the threads and the edges between nodes at different levels denote the communication corresponding to each pair of threads.

4.3.2 Optimization of objective function

The objective function developed in [KPT11] is used in the polymorphic scheduler in this work to decide on morphism for threads. The overall scheduler goal is to maximize the performance metric at the output (sink) threads in the TCG shown in Figure 4.2. This is because of the fact that an applications actual user satisfaction can be perceived only at the sink node. The computational relationship of the TCG also determines how many units of computational output of a thread $T_{i,j}$ are needed to generate one unit of computational output at the sink thread. For instance, 1 frame at the output of $T_{i,j}$ results in 1 frame at the output of the sink thread. Hence, the maximization problem of the performance metric $f(A_i)$ for the application

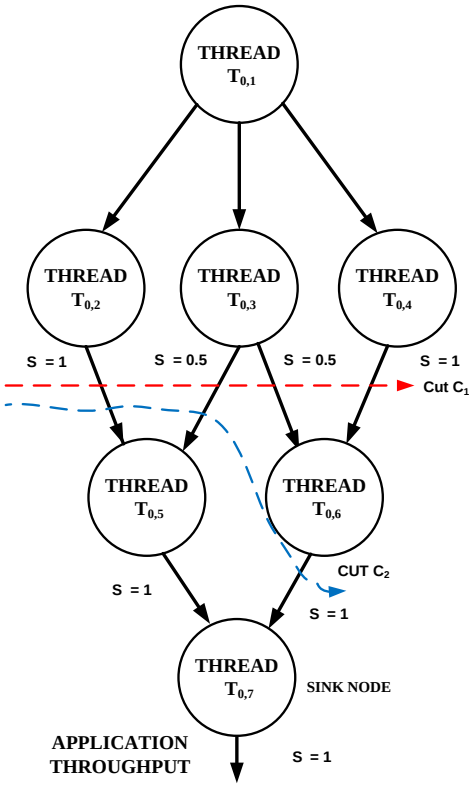

Figure 4.2: Thread Communication Graph

A_i is mapped into the local optimization problem at the thread $T_{i,j}$ as a scaling function $s_{i,j}$ times the original function $f(A_i)$. The local optimization function is to maximize $s_{i,j} * f(A_i)$. In the scenario of 1 frame at the output of $T_{i,j}$ resulting in 1 frame at the output of the sink thread, the scaling function $s_{i,j} = 1$. Often, this scaling relationship will be determined by the algorithmic choices within the threads, and not by their morphisms. What morphism will determine is the per unit time notion of the performance metric − not the scaling relationship. The contribution of a thread toward overall performance throughput depends on the active cut in the thread level flow graph − currently active threads.

Consider the TCG in Figure 4.2. Assume that the currently active threads are $T_{0,2}, T_{0,3}, T_{0,4}$ corresponding to the cut C_1. Let us consider how much performance throughput Thread $T_{0,2}$ contributes to application A_0. The scaling function is $s_{i,j} = 1$ for this edge. The scaling factors over all the edges in the cut C_1 add up to 2.5. Hence, $T_{0,2}$ contributes $\frac{s_{0,2}}{\sum_{j \in C_1} s_{0,j}} * f(T_{0,2}) = \frac{1}{3} * f(T_{0,2})$. On the other hand, at cut C_2, $T_{0,2}$'s contribution would be $\frac{f_{0,2}}{\sum_{j \in C_2} s_{0,j}} * f(T_{0,2}) =$

$\frac{1}{2.5} * f(T_{0,2})$. We call this contribution $f_{0,2,r}$ which is $T_{0,2}$'s contribution to the objective function for morphism r - this is captured in the output throughput $f(T_{0,2})$. The optimization problem is then stated as below:

$$Maximize\ S\ =\ \sum_{j=0}^{t_i} \sum_{r=0}^{r_{i,j}} f_{i,j,r} * Ready(T_{i,j}) * M_{i,j,r} \tag{4.2}$$

$Ready(T_{i,j})$ is 1 if the thread $T_{i,j}$ is in ready queue, and 0 otherwise. The scheduler has to select at most one morphism for $T_{i,j}$ by ensuring $\sum_{r=0}^{r_{i,j}} M_{i,j,r} \leq 1$ so that at most one $M_{i,j,r}$ corresponding to the selected morphism is 1. Hence the constraints are,

$$\sum_{r}\ M_{i,j,r}\ \leq\ 1,\ \forall i,j \tag{4.3}$$

$$\sum_{i,j,r}\ M_{i,j,r}\ R_{i,j,r}\ \leq\ (R_{cores} + R_{tiles}) \tag{4.4}$$

$M_{i,j,r} = 1$ if thread $T_{i,j}$ will assume morphism r in the current scheduling cycle. The second constraint ensures that the allocated software core and configurable core resources do not exceed the available system resources. The scheduler has complete information to compute $f_{i,j,r}$. Each thread in the active queue from Application A_i (identified through a group ID) contains its scaling function $s_{i,j}$ in its thread descriptor header. Thread descriptor for $T_{i,j,r}$ can also publish its throughput $f(T_{i,j,r})$. The $f_{i,j,r}$ for the objective function can then be computed by the scheduler as $\frac{s_{i,j}}{\sum_{T_{i,l} \in ReadyQueue} s_{i,l}} * f(T_{i,j,r})$.

4.3.3 Multi-Application Scheduling

There are many ways of combining the resource contention for multiple applications $A_0, A_1, \ldots, A_{n-1}$. For the ease of explanation, we will illustrate this with two applications A_0 and A_1 with t_0 and t_1 threads respectively. Let us assume performance model functions $S_0(A_0, C_S, C_C)$ and $S_1(A_1, C_S, C_C)$ to express performance metric for A_0 and A_1 as a function of software and configurable core resources C_S and C_C. Let $u_0(S_0)$ and $u_1(S_1)$ be the user satisfaction functions, as a function of their corresponding performance metrics. A simple, straightforward way to combine them would be to take a weighted average of the user satisfaction functions as $f(S_0, S_1) = w_0 * u_0(S_0) + w_1 * u_1(S_1)$. This function has the disadvantage since it is not

a normalized function, and hence during resource allocation the value of the currency keeps changing! Moreover, such a weighted average loses some of the desirable characteristics of the sigmoidal form.

An alternate way of resolving resource contention between these two applications is as follows. In the end, what we really care about is the marginal utility of an available resource such as a configurable core C to the user satisfaction, $\frac{\delta u_0}{\delta C_C}$ and $\frac{\delta u_1}{\delta C_C}$. Whichever application gives higher marginal utility in user satisfaction should be allocated the next available configurable core. Towards that goal, we can calculate:

$$U_0 = \frac{\delta u_0}{\delta S_0} * \frac{\delta S_0}{\delta C_C};\ U_1 = \frac{\delta u_1}{\delta S_1} * \frac{\delta S_1}{\delta C_C}$$

The nice part of sigmoidal functions u_0 and u_1 is that $\frac{\delta u_i}{\delta S_i}$ is just $u_i * (1 - u_i)$, an easy to compute function. The second component measuring sensitivity of the performance metric to the resources, $\frac{\delta S_i}{\delta C_C}$ can be easily approximated by considering each polymorphic thread in the *Ready Queue* belonging to Application A_i and averaging the throughput difference between its software and hardware morphisms: $\frac{\delta S_i}{\delta C_C} \approx \sum_{T_{i,j} \in ReadyQueue}(f(T_{i,j,c}) - f(T_{i,j,s}))$ where $f(T_{i,j,c})$ is the performance metric of hardware (configurable) morphism for thread $T_{i,j}$ and $f(T_{i,j,s})$ is the performance metric of software morphism for thread $T_{i,j}$.

4.4 Network Layer User Satisfaction

In mobile devices, multiple applications contend for limited resources in the underlying embedded system framework. Application resource requirements in mobile systems vary by computation needs, energy consumption and user interaction frequency. Quality of service (QoS) is the predominant metric of choice to manage resources among contending applications. Resource allocation policies to support static QoS for applications do not react to the changing demands of the user in contemporary network on chip (NoC) based embedded architectures. User satisfaction with the user interactions and user interface (UI) design ought to be the primary design driver. The application and operating system level user satisfaction research assumes that the throughput of inter-thread edges is limited only by the computational con-

straints of the nodes. With NoC, however, NoC resource allocation policies play an important role in determining the inter-thread communication flow's throughput and the resulting application level user satisfaction. We filter down the user satisfaction from an application layer attribute to a router level attribute to improve the resource and energy utilization for routing in order to leverage the user satisfaction at the application and system level.

We extend the application layer resource allocation based on user satisfaction model to the router resources in the NoC layer. We integrate NoC connection resources into application level user satisfaction by modeling a throttling factor, $0 < m \leq 1$, that limits the throughput of edges in the thread flow graph. The effect of NoC resource allocation (links, VCs, and other router resources) is to increase or decrease m for the corresponding edge of the thread communication graph resulting in optimization of the user satisfaction at network layer. In this work, VC allocation is implemented as a software layer at each router. An inbound header packet, at a router, requesting VC contains information about the throttling effect of previous routers in its path and the desired user satisfaction of the communication flow represented by the header packet. The VC allocator will increase or decrease m at the current router by accepting or rejecting the header packet.

In the case of router, communication flows from different threads of the application layer mapped to NoC nodes compete for virtual channels, buffers, switches and output ports. When there are two or more communication flows competing for network router resources, the user satisfaction model can guide the router control logic to allocate virtual channels for more than one flow. Resources can be allocated to multiple flows to maximize the user satisfaction as compared to traditional solutions that monotonically increase resources to a single flow without regard for its perceived effect on the user. Our methodology uses the sigmoid model in Figure 4.3 to develop VC allocation heuristic for QoS supported NoC.

Increasing router resources (for example, sharing VC with multiple flows) may improve the bandwidth from 0.6 flits/cycle to 0.7 flits/cycle and this effect proportionately improves the user satisfaction from 0.8 to 0.92 in the *linear* (Red lines) region of the sigmoid. However, allocating larger resources (for example, multiple VC to a single flow) may result in a larger bandwidth change in the *saturation* region (Blue lines) from 0.7 flits/cycle to 0.8 flits/cycle.

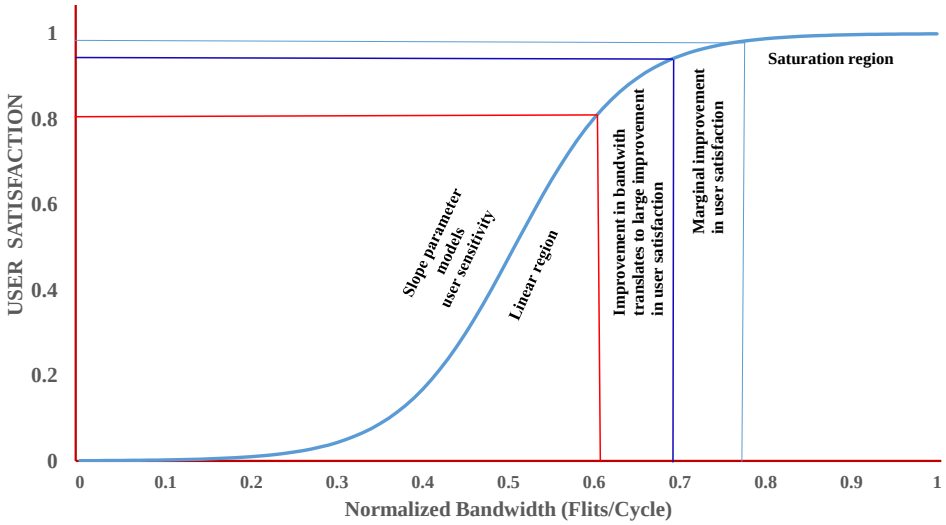

Figure 4.3: Sigmoid Function for NoC Router

However, this only gives a meager improvement in users satisfaction from 0.92 to 0.97. Modeling (or dynamically evaluating) the effect of resource allocation at NoC level on the performance parameter (like flow bandwidth) used in user satisfaction function is an interesting research problem addressed in this work. This strategy can be used for reallocation of router resources to flows that may be starved when competing flows are close to the saturation region of resource allocation. For example, consider two competing communication flows, F_1 and F_2, at a router which are requesting virtual channel allocation. The router calculates the user satisfaction (US) for flow F_1 from Figure 4.3 to be $US(F_1) = 0.75$. Allocating more VCs or buffers to F_1 will diminish the rate of increase (saturation region) of user satisfaction to $US(F_1) = 0.79$. However, for flow F_2, the initial satisfaction is $US(F_2) = 0.5$. Allocating extra VCs will result in $US(F_2) = 0.6$. The obvious decision is to allocate the router resources to F_2. Hence, the overall user satisfaction over all flows is maximized when user satisfaction gain per unit VC or buffer is maximized over all competing flows.

CHAPTER 5. POLYMORPHIC NETWORK ON CHIP ARCHITECTURE

Polymorphic network on chip (PolyNoC) [PZT13] is the simulation framework developed as a part of this research work for cycle accurate execution of polymorphic applications. In this chapter, a detailed description of the architecture and programming of PolyNoC is presented. The internal organization of PolyNoC is shown in Figure 5.1. The simulation environment is composed of five layers of abstraction. The *application layer* loads and executes user applications from memory. The application layer processes the data flow graph to generate threads for the scheduler to decide on the resource binding. The *polymorphic thread scheduler* layer advances the simulation by scheduling threads at each simulated clock cycle. Threads are queued up in C++ models of run and wait queues. Data structures keep track of free NOC resources, mapping of threads to resources, morphism of current threads. Threads can be preempted when there is a I/O event , communication event or timer interrupt. The *scheduling and mapping* layer evaluates the proposed sigmoid function model to calculate the marginal user satisfaction of individual threads that are being scheduled in the current cycle. Communication among threads generates data packets. Data stored in the buffers of FPGA morphism or the CPU software thread buffer is converted into packets for NOC transmission by the *packet management* layer. Packet buffers are allocated for headers and initialized with thread identifier information, origin clock cycle etc. Packets are then injected into the NOC *IP_block* source buffers for arbitration and transmission.

The PolyNoC simulator [PZT13] is parameterized to model a M row X N column mesh communication network shown in Figure 1.1. Each grid point in the mesh network is the location of a resource such as CPU, programmable FPGA cells, input-output(IO) controllers or other computing cores.

IP_block is the simulation primitive used to model a resource. Each resource is interfaced with the communication bus through a router.

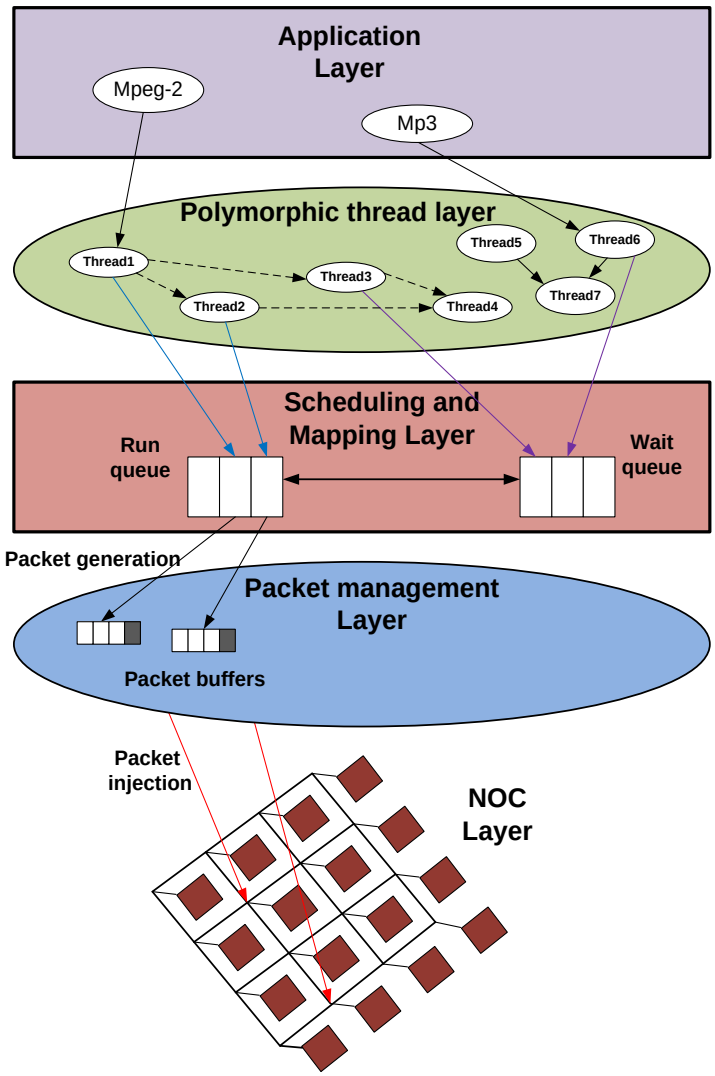

Figure 5.1: PolyNoC Simulation Framework

Each link between neighboring *IP_block* is bi-directional in the North, South, East and West directions. First in first out(FIFO) buffers on each bi-directional link hold data packets in that specific direction. The size of a FIFO buffer is configurable in the simulator during initialization. A high level model of our router is shown in Figure 5.2.

Each bi-directional link has input and output FIFO buffers to store and forward packets to their final destinations. Source and sink FIFO buffers interface the *IP_block* with the router and

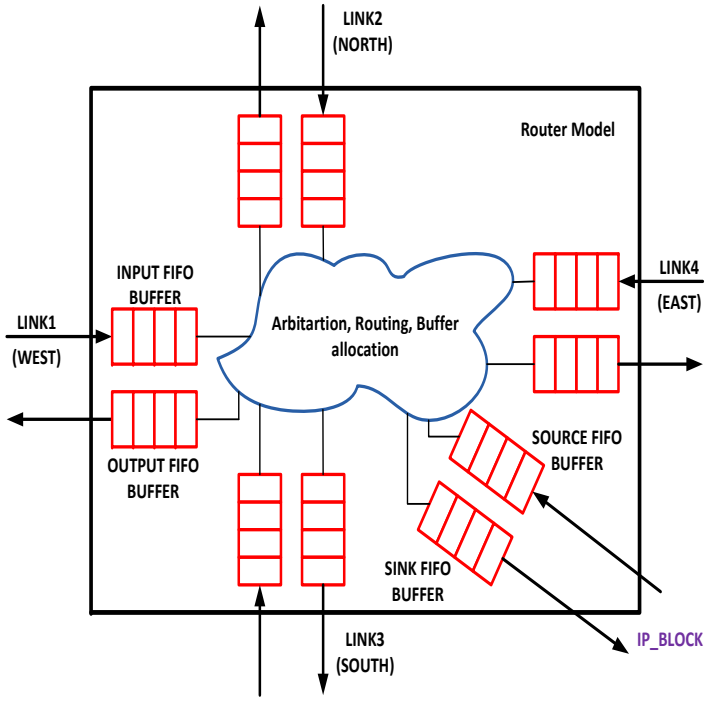

Figure 5.2: Router abstraction in the simulator

thus connect the resource with the NOC mesh. Header of data and control packets arriving at the input FIFO buffers is decoded by the router to decide allocation of virtual channels (VC) at the input FIFO buffer. VCs arbitrate for a slot in the output FIFO buffer of the requested link direction in round robin manner. Packets which have arrived at the destination router are placed in the sink buffer for further processing by the *IP_block*. Similarly, packets originate from *IP_block* intending to communicate with another *IP_block* and begin their journey in the source FIFO buffer. Worm hole switching is employed to reduce buffering needs at each router. The number of flits per packet and the number of VCs in the router is also configurable at initialization of the simulator.

Threads are minimum granularity computation units in this framework. Messages are passed between communicating threads to implement the data flow graph of an application. The binding of a computation unit to a set of computing resource in the NOC mesh is determined by the scheduler dynamically subject to resource availability and supported thread morphisms.

Packets from sender thread are stamped with a uniquely generated thread identifier. This is useful for updating operating system data structures when the packet is received at the destination. For example, time stamp could be used to wake up a waiting thread upon the arrival of data.

5.1 PolyNoC Simulator Layers

5.1.1 NoC Layer

NoC layer class defines various parameters of router, link and policies to manage routing, arbitration. Table 5.2 lists the members of network layer and their definitions. NoC layer interface functions defined in *nw.cpp* of the PolyNoC simulator are used to send/receive packets, check status of virtual channels, arbitrate for crossbar etc.

5.1.2 Packet Management Layer

Packet parameters such as header size, payload size, data width can be varied to estimate the dependence of performance and power consumption. Table 5.1 lists the members of packet class and their definitions. Interface functions defined in the PolyNoC simulator file *packet.cpp* are used by higher layers to insert, queue and retrieve packets from the NoC layer of Figure 5.1.

Table 5.1: Packet Class

Member	Definition
dest_x, dest_y	Destination coords of packet
origin	origination NoC node
src_tid	source thread identifier
dst_tid	destination thread identifier
data	packet buffer
data_size	packet payload size
origin_cycle	packet creation cycle
insert_cycle	insert packet at later time
reached_cycle	packet reached destination cycle
reached_dest	flag indicating packet is in transit
pkt_delay	number of cycles from source to destination
packet_type	header = 0, tail = 1, data = 2
stream_id	communication flow identifier

Table 5.2: Network Class

Member	Definition
nbrs	neighbours of NoC node
x,y	coords. of NoC node
send_flag	NoC node ready to transmit
dirs	NoC node bi-directional links
o_buf	NoC node output buffers
l_buffer	Physical channel buffer
vcs	Virtual channels
thread	thread executing in NoC node

5.1.3 Scheduling & Mapping Layer

Scheduling layer manages the queues which hold application threads. Run queues keep track of polymorphic threads currently ready to run when NoC resources become available. Threads which are waiting for I/O or blocked due to data/control flow precedence are held in the wait queue. Table 5.3 lists the members of scheduling & mapping class and their definitions. Interface functions defined in the PolyNoC simulator file *scheduler.cpp* are used by higher layers to request for NoC compute resources, check status of running threads, mapping of threads to NoC resources etc.

Table 5.3: Scheduler Class

Member	Definition
tile	NoC node type
run_state	NoC node is busy
tid	thread identifier on NoC node
run_list	threads ready to run
wait_list	threads in wait list
running	threads executing currently
threads_all	all threads in the system

5.1.4 Polymorphic Scheduling Complexity

The scheduler maintains data structure per thread to determine feasible threads which can be scheduled during each scheduling epoch subject to resource constraints. The master data structure, *polymorphic thread table*, maintains the scaling factor corresponding to different

cuts a thread may belong to. A snapshot of the data structure is shown in Figure 5.3. The polymorphic thread table is indexed using the thread identifier which is unique for each thread created or forked. The scaling factor for each thread depends on the cut to which the thread belongs to during the life time of the application. There are two cuts, C_1 and C_2, to which thread, $T_{0,2}$, can belong during application life time. Each cut is associated with a scaling factor which is also found in the polymorphic thread table. The *morphism table* contains information about the thread morphism choices, resource requirements per morphism and the associated user satisfaction. A pointer from the polymorphic thread table indexes into the morphism table to determine the user satisfaction for current resource availability.

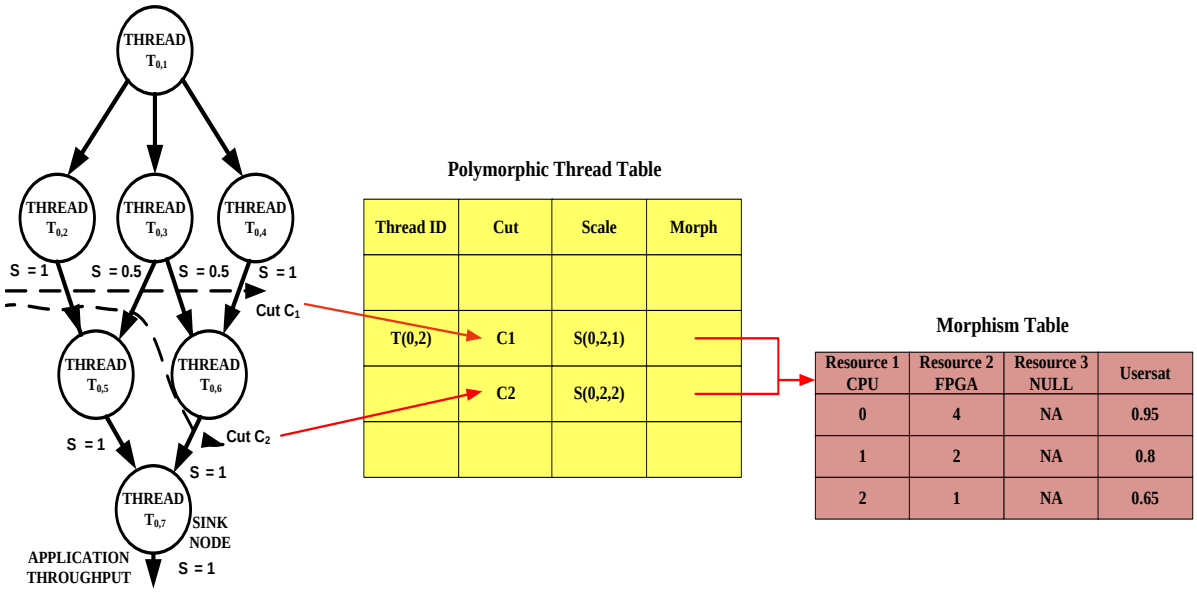

Figure 5.3: Scheduler Thread Data Structures

The two compute cores considered in this thesis are CPU and FPGA. The morphism table shows different user satisfaction levels for various units of CPU and FPGA allocated to thread $T_{0,2}$. Hence, the polymorphic thread table and morphism table contain all the information in the scheduling phase to compute overall user satisfaction for applications in Equation 4.2.

The user satisfaction model in Section 4.1 is continuous for the variable t which models parameter such as application throughput, router flit rate etc. In reality, compute resources are discrete and user satisfaction is not continuous as resources are varied. This fact enables

the use of tables to store the pre-determined user satisfaction for morphisms instead of computing values for sigmoid function at various parameter settings. The heuristics developed in [KPT11] evaluates indexing strategies into the morphism table for initial allocation of resources to determine feasible threads for scheduling. The morphism is changed only in steps to minimize scheduling overhead. The thread's morphism will converge to the morphism which will maximize the overall user satisfaction using the strategy of marginally increasing or decreasing the index into the morphism table.

The polymorphic scheduler chooses threads which maximizes the user satisfaction during each scheduling cycle. The set of threads which provide most marginal increase in user satisfaction for incremental resource allocated will add to the overall user satisfaction. The allowable morphism change in morphism table is only to the next higher morphism if the heuristic is to initialize all threads to the lowest morphism when forked or created. Hence, ready threads can be inserted into a red-black tree similar to that of Linux CFS [CFS] using marginal increase in user satisfaction as the key. The ordered polymorphic threads based on increasing marginal user satisfaction can be chosen in O(log N) time by visiting k left most nodes. The k polymorphic threads then maximize user satisfaction using the allocated compute cores in that scheduling cycle. This makes polymorphic scheduler comparable to Linux CFS for the time to choose ready to run polymorphic threads from the run queue.

5.1.5 Polymorphic thread Layer

Polymorphic thread layer manages the thread flow graph of the application. Threads are created and destroyed as specified in the thread flow graph. Data structures keep account of parent, child, exit status of child etc. Threads can be forked and joined in high level C++ code. This layer assign thread identifier to newly created threads. Table 5.4 lists the members of scheduling & mapping class and their definitions. Interface functions defined in the PolyNoC simulator file *appthread.cpp* are used by higher layers to fork/join threads, get thread identifier of child/parent thread, status of execution of child/parent thread etc.

Table 5.4: Polymorphic thread Class

Member	Definition
parent	thread identifier of parent
child	thread identifier of recent child
child_all	thread identifiers of all children
resource	thread NoC node type
resume_thread	waiting for NoC resource
wake_up	control/data dependency is satisfied
exec_join	join with parent thread
wait_for_receive	waiting for data dependency

5.1.6 Application Layer

Application layer interfaces with the external world by reading application binary and loading into memory. Parameters for applications are read from command line or an input file. The command line or input file is parsed for initializing simulator components. Output can be written to file or displayed interactively to see the progress of packets in the network and threads completion statistics is dumped.

5.2 PolyNoC simulator API

The simulator is completely written in C++ to facilitate overriding of available API functions by the user code. Major functional blocks of the NOC style interconnect architecture are abstracted using class definitions. A class is defined for each of IP_block, Router, Buffer, Links, Scheduler, Packet and Application thread. Linux style thread API and OS variable names are mimicked and implemented to provide minimal overhead for porting existing multi-threaded applications to the polymorphic simulator.

The application thread class, *app_thread*, is the workhorse for writing user code. A user application thread is created by inheritance of the *app_thread* base class. The derived class needs to override the virtual function *operator()()* to implement the desired thread functionality. In Figure 5.4, *operator()()* is overridden by the derived class, *my_thread1*, to print the thread identifier. The built in API function , *get_tid()*, can be used to get the thread identifier of the currently executing thread. Each thread is assigned a unique identifier upon creation.

```
class my_thread1 : public app_thread
{
private:  // local variables of threads
 int count;

public :
 my_thread1();

 virtual void operator()() ;  //thread implementation
};

void my_thread1::operator()()
{
cout << "this is my_thread1"<<endl;
cout << " my_thread1 id is "<< get_tid() << endl;
}
```

Figure 5.4: Creation of new polymorphic thread

Figure 5.5 illustrates the private variables of the base class definition for application thread in the polymorphic simulator.

A host of state information is maintained in the *app_thread* class. *get* and *set* functions are available to access the state variables. For example, *get_mapped()* returns true if the invoking thread is mapped to an IP_block. *set_mapped()* is used by the polymorphic scheduler to mark the thread as mapped if IP_block can be allocated to it. However, some *set* functions may be used by the simulation environment and hence, are not accessible by the user code. *tile* variable holds the IP_block address in the NOC mesh to which the thread is mapped. *run_state* is the current executing state of the thread - 1) running, 2) ready to run, 3) wait for resources. The variables *resume_thread*, *wake_up*, *exec_join*, *wait_for_receive* are modified by the scheduler layer of the simulation environment to manage multiple threads running in the system. Each thread may or may not be polymorphic depending on the user application. The variable *morphism* identifies the current morphism assigned to the thread by the scheduler.

The next example in Figure 5.6 illustrates the *fork* and *join* constructs of the polymorphic simulator. An application thread object has to be passed as the argument to the *fork* function.

```
class app_thread
{
private :
 int mapped ;
 int tile;
 int run_state;
 int tid;
 int fid; //father id of this thread

 int done;
 int die;

 list <app_thread *> child; //childs of this thread which are active
 list <app_thread *> child_all; //all childs of this thread

 app_thread *parent; //parent of this thread

 list <nw_int *> resource_history; //all nodes where the thread ran, prempted etc

 nw_int *resource; //thread is running on this node
 int node_num; //node number where current thread is running

 int resume_thread;
 int wake_up;
 int exec_join;
 int wait_for_receive;
```

Figure 5.5: Application thread base class

Thread 2, *my_thread2* class, forks the child Thread 1, *my_thread1* class, and waits for the child to complete at the *join* statement. The *fork* function takes the object *child* as an argument. Upon the execution of the fork statement, the simulation environment assigns a thread identifier to the child thread and adds the thread to the run queue if resources are available. The child thread can get the parent thread identifier using the *get_fid()* call. Communication between threads is supported by using *send* and *receive* primitives. The thread identifier of the receiver thread is passed as argument to the *send* function. Data to be shared with the receiver thread is stored in a buffer and passed as the second argument. *send* primitive is implemented in two flavors - blocking and non-blocking. Blocking send suspends the sender thread until the receiver thread finishes processing and the thread is placed in the wait queue. Non blocking send continues execution past the send statement without waiting for the receiver to finish. The *receive* function is blocking only. The data in the buffer is enclosed in one or more packets with header and footer information for routing in the NOC mesh.

```
void my_thread1::operator()()
{

  cout << "this is my_thread1"<<endl;
  cout << " my_thread1 id is "<< get_tid() << endl;

  cout << " my_thread1 parent id is "<< get_fid()
  << endl;

}

void my_thread2::operator()()
{

 my_thread1 child;

 cout << "this is my_thread2"<<endl;
 cout << " my_thread2 id is "<< get_tid() << endl;

 fork(&child);

 join();

}
```

Figure 5.6: Fork and join example

Scheduler class, *scheduler*, manages various thread queues, mapping of threads to NOC resources and choosing morphisms of threads using the sigmoid model for user satisfaction. This class maintains a list of all active threads in the system. A glimpse of different APIs used by the scheduler can be seen in the code excerpt of the simulator loop in Figure 5.7.

The simulation environment is initialized by invoking the *init()* function of the object, *polysched* instance of the scheduler class. This function creates all the queue objects, thread resource mapping tables and morphism state objects. The user can also provide applications to be invoked before the start of the simulation run. This is done using the *exec()* call and passing the thread object. In this example, *app1* and *app2* are two applications to be started before the simulation run. The call to *schedule_threads* invokes the scheduler to analyze various queues, perform resource allocations, decide morphism changes among other functions. Several other events such as communication primitives (send and receive), waiting for I/O can invoke scheduler before the time quantum of a thread expires.

Packet repository class handles the creation, manipulation and deletion of packets. Packets created are registered with the packet repository manager, *packet_repos* instance in the example,

```
polysched.init(); // init resources and other stuff which
                  // cannot be done by the constructor

polysched.exec(&app1, &app2); //start initial set of applications

//grand simulation  loop
while(polysched.end_simulation() || packet_repos.end_simulation())
 {
        cout << "Cycle " << cycle << endl;

        polysched.schedule_threads(cycle);

      for(int i = 0 ; i < noc_sz * noc_sz ; i ++)
        {
           thr = (*nw_mesh[i]).get_thread();

           if(thr != NULL)
                   (*nw_mesh[i]).run_thread();

           (*nw_mesh[i]).print_buffers();
             (*nw_mesh[i]).routing();
             (*nw_mesh[i]).print_buffers();
        }

 cycle++;
 }
```

Figure 5.7: Scheduler API example

for tracking purposes and memory management. Information about source IP_block , destination IP_block , origination cycle, path of the packet in the NOC etc. can be accessed using built in API. Finally, the simulation is terminated by call to *end_simulation* in both the scheduler and packet repository. Calling *end_simulation* ensures that all threads have completed execution and all packet have reached their destinations.

5.3 Polymorphic Applications

MPEG-2 video standard was developed to address bandwidth reduction for transmission of television quality digital video. It is a suite of audio and video codec based on lossy compression techniques. Coding and decoding algorithms of the video standard operate on 64 pixels of image data organized as a 8 X 8 matrix. The algorithms involved in the decoding of a video file are shown in Figure 5.8.

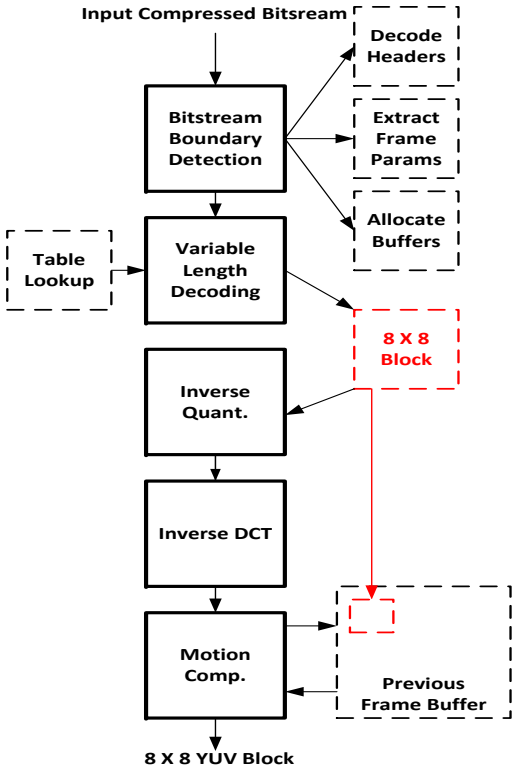

Figure 5.8: MPEG-2 Video decoding pipeline

The input bit stream from a video file is parsed for the occurrence of a unique start code indicating the start of a valid mpeg-2 stream. A FIFO buffer serves as a bit reservoir for the input bit stream. Bits are sourced by the different steps in Figure 5.8 from the reservoir. Global parameters of the video file are first extracted. Decoding proceeds by detecting *group of pictures*(GOP) start code and the GOP header is analyzed to retrieve time stamp information and other flags. MPEG-2 stream may consist of multiple GOP frames. GOP frames can be decoupled into multiple individual units for decoding. All the video samples used in our simulation have multiple GOP frames. Picture, Slice and Macro block headers are the next three levels of hierarchy shown in Figure 5.9.

The variable length of the codes used to decode the motion vectors and AC/DC co-efficients imposes serial processing of subsequent codes. Access to code tables for decoding multiple GOP frames creates contention. Picture type is determined after decoding the picture header.

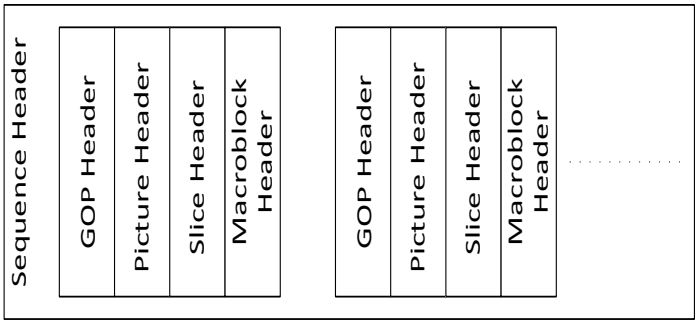

Figure 5.9: MPEG-2 Header hierarchy

MPEG-2 standard supports three picture types - I type, P type and B type. I type picture does not have any temporal redundancy and hence requires more encoding bits. It can also be decoded independently. There are more accesses to table lookup when compared to computations in I type picture. P type pictures are predicted based on an earlier I type or P type. More redundancy results in lower bits of encoding and lower table look up with higher computation needs. B type pictures are predicted based on previous and subsequent I and/or P type pictures. For example, a sample stream of pictures in MPEG-2 may have the sequence I, P, P, B, P, I, P, P, P. I type pictures in the picture stream also can be decoupled for individual decoding. Decoupling the stream results in two individual clusters of picture streams - 1) I, P, P , B, P and 2) I, P, P, P. There is interdependency between P, B pictures on I and/or P pictures for decoding. Pictures are further divided into slices and slices are subdivided into macro blocks. Each slice can be decoded in parallel. Observation in this section are used for the multi-threaded implementation of MPEG-2 decoder. Table 5.5 lists the primary classes and some interface functions used in our C++ MPEG-2 implementation. The classes in Table 5.5 encapsulate commonly used MPEG-2 computational functions. The user can change the implementation to change the morphism characteristics. We use two morphisms for computational threads which have well known CPU and FPGA implementations. Kernels such as inverse discrete cosine transform (IDCT), table look up, motion vector compensation have several FPGA implementations in research literature.

Table 5.5: MPEG-2 Polymorphic implementation

Class	Interface functions		Morphism
mpeg2_init	mpeg2_init_params	initialize buffers	CPU
mpeg2_params	get_vertical_size	video vertical size	CPU
	get_horizontal_size	video horizontal size	
	startcode	detect start code	
	getseqhdr	get sequence header	
	getgophdr	get GOP header	
	get_pict_type	picture type (I,P,B)	
mpeg2_main	mpeg2_run	invoke MPEG-2 application	
	mpeg2_gop	decouple GOP frames	CPU
decode_picture	decode_pict_init	Frame buffer initialization	CPU, FPGA
	sync_buffers	Frame buffer updating	CPU, FPGA
	motion_vectors	motion vector compensation	CPU, FPGA
	get_MBs	Macro block decode	CPU, FPGA
idct	idct_compute	compute IDCT of 8x8 block	CPU, FPGA
mpeg_buf	mpg_fill_buffer	Fill bit reservoir	CPU, FPGA
	mpg_peek_buffer	Look ahead into reservoir	CPU, FPGA
	mpg_readbits_buffer	Read from reservoir	CPU, FPGA

Our second application is a C++ implementation of the MP3 audio decoding standard. MP3 coded bits are also divided into multiple physical frames of data. Physical frame size is fixed and each physical frame is preceded by a *sync* code and *side channel information*. Physical frame boundaries can be detected by scanning for the sync code in the bit stream. However, MP3 frames are also viewed as logical frames for achieving variable bit rate. Physical frames and logical frames may not always coincide. A logical frame may span more than one preceding physical frame. This means that data of the current physical frame may be present in one or more previous physical frames. Hence, even though individual physical frame's side channel information can be decoded, there may be dependency between successive physical frame data. We compute the starting index of logical frame to initialize buffers. All threads in MP3 decoder use CPU morphism except the modified inverse discrete cosine transform (IMDCT) computation. IMDCT uses CPU and FPGA morphisms.

5.4 Experiments & Results

The experiments in this section are aimed at demonstrating two components of PolyNoC - 1) application simulation in PolyNoC framework and 2) compare the scheduling performance of sigmoid based user satisfaction model with traditional scheduling methods. Two multimedia applications, namely MPEG-2 decoder and MP3 decoder are ported to PolyNoC framework as case studies for polymorphic thread model. The MPEG-2 decoder C source code was developed by MPEG Software Simulation Group. We have ported the whole decoder to C++ from scratch with polymorphic simulator as the final target. MP3 decoder is also implemented in C++ from an open source C code.

The source thread of MPEG-2 application in Figure 5.10 reads video files from the disk and parses for the start code. It creates bit reservoirs for GOP(group of pictures). The source thread decodes the global parameters such as height, width, chroma format, output format and other details of the video file. The values are stored in a global class which is declared as a friend class for access of these data from other classes in the simulation. Temporary buffers are allocated to store GOP coded bits of video file.

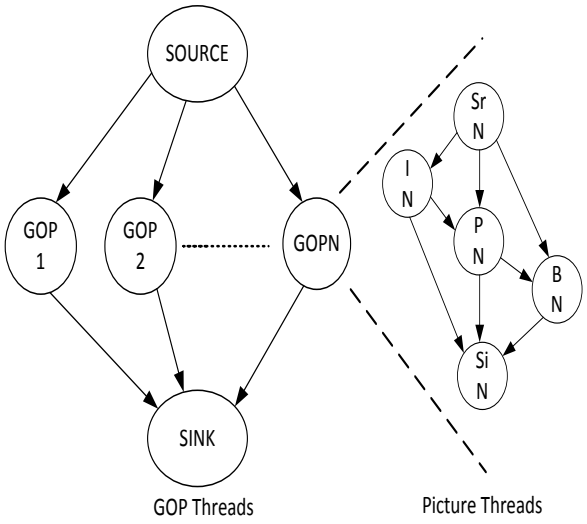

Figure 5.10: MPEG-2 Polymorphic threads abstraction

Each GOP frame is assigned to a single thread for decoding. The GOP headers are decoded by the GOP thread and other bit level processing is done to retrieve the coded bits for individual picture frames. The gop thread now forks picture threads for each picture type present in the picture stream as shown in Figure 5.10.

Decoding of coded coefficients and motion compensation differences involves table look up of Huffman codes. FPGA LUT and On Chip RAM are efficient for table lookup and this can be done in 1 or 2 cycles. To avoid contention from several GOP threads to a single On Chip RAM for table lookup, the tables are copied to the morphism of each GOP thread. Research literature has plethora of efficient lookup table implementations such as multi-cycle lookup to reduce storage requirements. The implementation and cycle length can be configured in the MPEG-2 GOP thread. Each *GOP* is spawned into a polymorphic thread which is scheduled on the NOC resources based of user satisfaction and resource availability. The granularity of polymorphic threads can be further modified to generate threads at the *picture, slice* or *macro block* level.

The physical attributes of the four video files used in our simulation are listed in Table 5.6.

Table 5.6: Video file attributes

Video file	Size	Max. frame rate	# of GOP	Avg. Picture frames
tennis.mpg(M_1)	352 x 240	30 fps	22	6
football.mpg(M_2)	352 x 240	30 fps	3	12
garden.mpg(M_3)	352 x 240	24 fps	22	6
salesm.mpg(M_4)	352 x 240	24 fps	6	6

The applications discussed in this section are used for evaluating the performance of user satisfaction based thread scheduling over traditional first come first serve (FCFS) and earliest deadline first (EDF) scheduling algorithms. Applications can be launched at random times in the simulation. However, in real life situations, a user may launch multiple applications successively or web browsers can spawn several video decoding threads for multimedia contents simultaneously. Three scenarios are used to generate application traffic as shown in Figure 5.11.

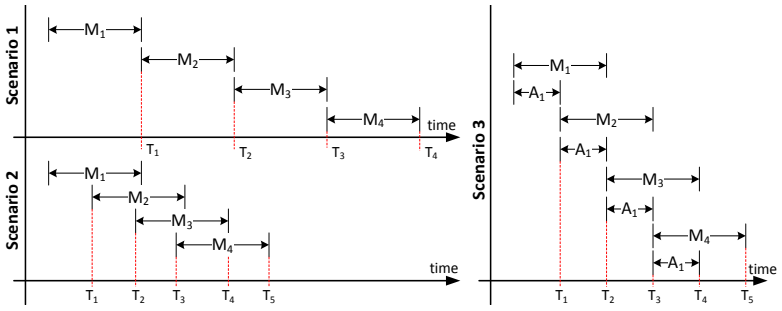

Figure 5.11: Application traffic scenarios

In Scenario 1, each of the four video files in Table 5.6 are decoded successively after the prior decoding is complete. This is the base case for comparing the user satisfaction based scheduling efficiency. Only threads from a single video decoding application are present in the simulation at any time in Scenario 1. All the resources of the NOC are available without any contention. In this case, loss of user satisfaction is primarily due to network inefficiencies in routing, arbitration and buffer allocation in the NOC. Multiple video file decoding are overlapped in Scenario 2. Successive video decode is invoked approximately midway during the decode of the prior video. This results in contention for NOC resources from threads of more than one video decode application. In Scenario 3, an audio decode application (A_1) is invoked along with each video decode application. Audio application is invoked a total of four times. In this scenario, resource contention occurs at the start of the simulation. The audio application however is light weight when compared to video decode.

The simulation is run on a 4 X 4 NOC with 6 CPU and 10 FPGA resources. Each CPU is modeled to execute two software threads. Threads ready to be scheduled, in the run queue, are allocated to the NOC resources using one of the three policies - 1) a set of threads which increase the marginal user satisfaction (US) are scheduled on free resources, 2) one or more threads at the head of the run queue are scheduled ($FCFS$), 3) one or more threads with earliest deadline in increasing order is chosen to be schedule in earliest deadline first (EDF). The three scenarios of Figure 5.11 are executed in the simulation to generate polymorphic threads and packet traffic in the NOC. User satisfaction is averaged over all executed threads and reported

at time instants T_1, T_2, T_3, T_4. Figure 5.12 plots the average user satisfaction for scenario 1 for each of the three scheduling policies.

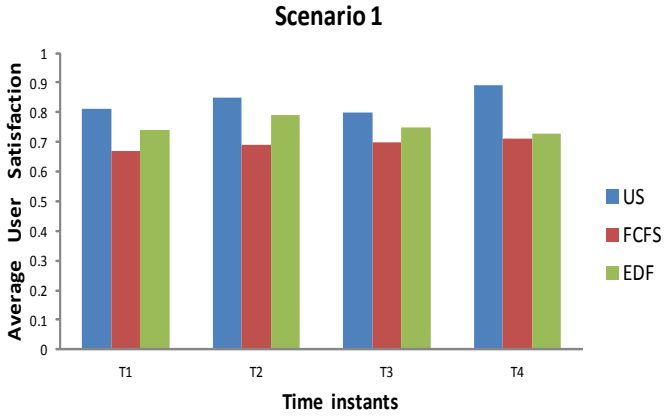

Figure 5.12: Average user satisfaction for Scenario 1

The average user satisfaction is better as compared to FCFS and EDF when morphism is chosen by the polymorphic scheduler to increase the marginal user satisfaction (*US)* as seen in Figure 5.12. The best morphism is statically chosen for a thread when scheduling using FCFS and EDF policies. EDF policy does perform better than FCFS since threads in the run queue with fast approaching deadlines are chosen for resource allocation and these happen to be I picture threads which have higher marginal user satisfaction. However, dynamic morphism transitions are allowed in *US* scheduling policy which utilizes CPU morphism for I threads if FPGA morphism change is not possible. Hence, EDF policy does not utilize free resources for the most gain in user satisfaction. Figure 5.13 plots the results for average user satisfaction resulting from overlapped execution of video decode applications. Contention for FPGA morphism from threads of multiple decode applications forces the polymorphic scheduler to choose CPU morphism for many more threads as compared to Scenario 1. Note the lower user satisfaction when two video decode applications are executing in the simulation as compared to Scenario 1. However, EDF and FCFS scheduling policies suffer a much larger drop in user satisfaction. This is because of the fixed morphisms of threads. Statically fixed FPGA morphism for *I threads* results in these thread suffering much longer wait times in the wait

queue for FPGA resource to free up. Also, note that this is specific to the 4 X 4 NOC with fixed number of CPU and FPGA resources. It is possible to improve the user satisfaction either by increasing the number of FPGA resources or the size of the NOC. Thus, our tool enables exploration of such design points give a set of multimedia applications.

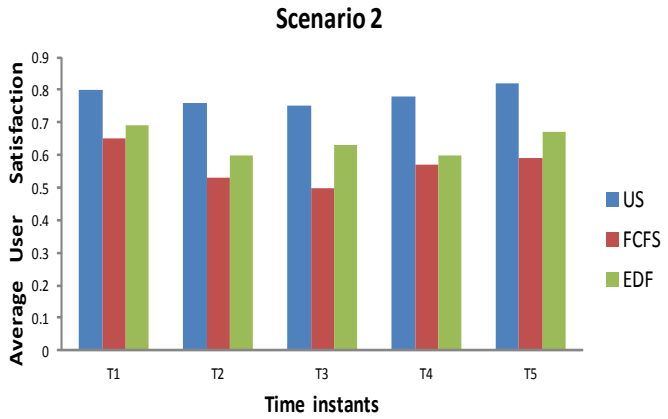

Figure 5.13: Average user satisfaction for Scenario 2

The third scenario generates threads by executing three simultaneous applications in our simulator. The audio decode application consumes more CPU resources than FPGA resources since we designed only the IMDCT thread to be polymorphic. The lower user satisfaction is due to higher contention for CPU resources from the audio decode threads in the case of polymorphic scheduling policy as seen in Figure 5.14.

Video decode threads have to compete for CPU resources with audio threads for morphism transition when FPGA resources are busy. FCFS and EDF policies suffer for the same reasons as in Scenario 2. The average user satisfaction is lower when the number of applications increases as seen from the above three scenarios. However, polymorphic scheduling optimizes user satisfaction more effectively in comparison to EDF and FCFS policies in all three scenarios. Polymorphism enables the scheduler to choose the next best morphism using late binding of computation to resources when resource allocation for the best morphism is not feasible. This section described the details of the architecture and implementation of a simulation tool for exploring polymorphic thread design space in an NOC communication architecture. We have

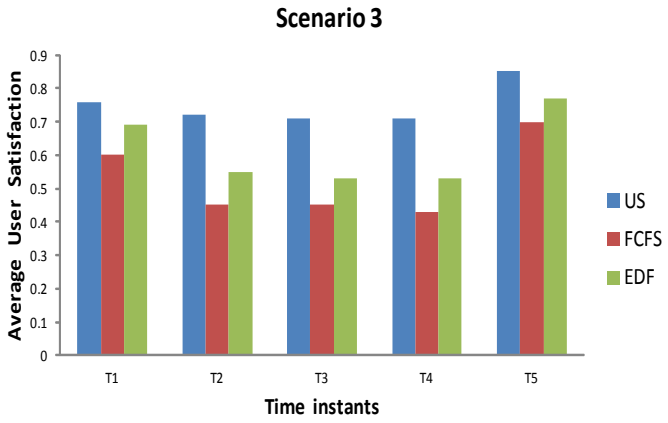

Figure 5.14: Average user satisfaction for Scenario 3

implemented polymorphic multimedia applications to compare the effectiveness of user satisfaction scheduling policy against traditional FCFS and EDF policies. The results demonstrate that polymorphism is an effective way to utilize the NOC resources during contention from multiple threads. NOC routers are passive in the current simulator model.

CHAPTER 6. USER SATISFACTION MODEL FOR NoC ROUTER

Modern mobile devices deploy rich and complex systems on chip (SoC) that include or are likely to include NoC [DT01, BM06] for inter-core communication. For fine-grain threading, NoC bound communication traffic represents the critical edges in application TCG. Hence, NoC performance is also strongly coupled with user satisfaction at the communication layer. User satisfaction can be asymmetric along multiple applications, and within an application it can be saturating. However, at NoC layer, we tend to allocate resources to improve overall bandwidth or latency until resources such as virtual channels (VC) are exhausted. To enable permeation of the user satisfaction model into the NoC layer [PT13], the research in this chapter has addressed the following two issues – (1) translate user satisfaction metric at application level into user satisfaction metrics at NoC layer (2) develop routing heuristics to use the user satisfaction metrics at NoC layer. The focus of this chapter is in exploring a holistic user satisfaction optimization at the NoC layer. We make three contributions in this chapter. First, we propose to use the user satisfaction model in Section 4.4 to measure and optimize the utilization of router buffers using virtual channel abstraction. Second, we describe the modifications required for virtual channel allocation to support the user satisfaction based scheme. Third, we develop a simple energy model for thread communication in the NoC and apply user satisfaction to optimize energy utilization for a critical set of applications.

Application layer uses the services of the NoC layer. The user satisfaction model provides a unified *QoS like* metric for design optimization at the application layer and the network layer. Most prior efforts reviewed in Section 2.2 related to virtual channel allocation have limited the focus exclusively at the NoC layer ignoring the effects of such solutions on the higher level abstraction layers such as the application layer. Our *QoS like* metric extends seamlessly to the network layer from the application layer. As far as we know, our work is the first attempt

to integrate user model into the NoC layer resource optimization and energy modeling for communication flows.

6.1 User satisfaction metric for VC allocation - Motivation

Virtual channel allocation stage of the control logic allocates buffer space for incoming packets in the router. VC is allocated by the head of the line or header (HOL) packet that contains control information about the communication flow. VC allocation is also typically the first stage in the control logic and very critical in the completion of the communication circuit from source to destination. Unavailability of VC results in blocking of the HOL packet [HOM07]. Similarly, arbitration (ARB) for the switch to output port constitutes the second stage on the critical path. Several works have also investigated techniques to increase bandwidth by manipulation of VC allocation. VC is also important from another aspect - QoS (quality of service). Quality of service is an important parameter to measure the overall performance of the SoC from the end user point of view. Metrics such as response time for interactive applications, frame rate for video applications serve as QoS metrics for embedded systems. Most work ignore modeling of QoS from the end user view in the NoC abstraction. Routers effect is to throttle the maximum available bandwidth for a single communication flow due to the presence of multiple competing flows. In this chapter, we model the router bandwidth throttling and incorporate it into the user satisfaction model of Figure 4.3 using the *slope and throttle parameter* (See Section 6.3) of the curve.

To motivate the contribution of this work, we will provide a simple example shown in Figure 6.1. Consider two applications - a video and a web browser application running on the 3X3 NoC. Each application has two communicating threads which are mapped to the nodes of the NoC infrastructure. Figure 6.1 depicts the first two columns of the 3X3 NoC with the routers being blown up in size to show the communication routes. The VLC thread is mapped to *Node 0* and the ME thread is mapped to *Node 4* for the video application. Similarly, the W1 thread is mapped to *Node 6* and the W2 thread is mapped to *Node 1*.

The *header*, *data* and *tail* packets of both applications compete for buffers, virtual channels and output port at each router until the destination node is reached. Packets (Red colored)

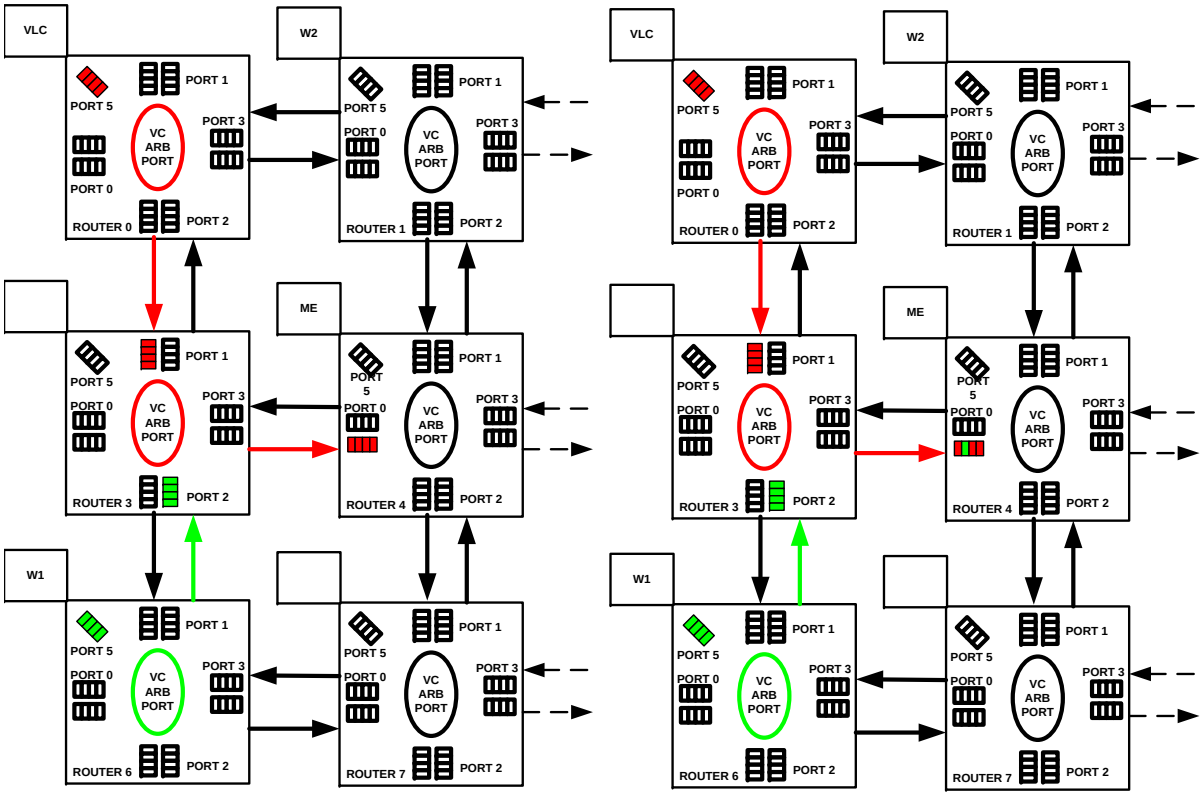

Figure 6.1: Two applications communicating in NoC

originating from the VLC thread trace the Red colored path through the network to the ME thread as seen in Figure 6.1(a). W1 thread of the web browser application sends packets (Green packets) by the Green route to W2 thread.

In this example, the flow from VLC→ME (shown using Red colored router elements) and W1→W2 (shown using Green colored router elements) compete for buffer resources at the input ports of Router 2. Depending on the VC allocation policy and arbitration protocol, one of the competing data flows will experience HOL blocking at Port 3 of Router 2 (Figure 6.1(a)). It may be random for round robin policy. If QoS policy is implemented, video application may have guaranteed QoS due to decoding constraints. In this case, HOL blocking will occur for data communication flow for the web browser threads. However, once a minimum QoS is achieved for the video application data flow, attempts can be made to reallocate the excess

bandwidth to other blocked HOL packets in the router. This is the crux of our technique which is to allocate VC and arbitrate for outport based on optimizing the bandwidth of multiple communication flows simulataneously. The flow with maximum marginal gain in user satisfaction receives the next VC or buffer. Once the bandwidth needs of a single flow are satisfied by its user satisfaction moving to saturated region of Figure 4.3, it is not likely to receive additional NoC resources. The excess bandwidth over the guaranteed QoS can now be reallocated to other communication flows. The saturation region of the sigmoid models any resource allocation above the guaranteed QoS. Figure 6.1(b) depicts the state of buffers in Router 4 after reallocation of VC to communication flow W1→W2. A Green packet of flow W1→W2 gets to share the VC allocated to the flow VLC→ME. In the user satisfaction model, physical parameters such as bandwidth, throughput are mapped to users perceived satisfaction. We attempt to use user satisfaction as the QoS metric to perform VC allocation as discussed in the work. Our argument is that optimizing NoC physical constraints do not directly map into the optimization of top level application abstraction. This is because decisions for mapping and allocation of NoC nodes to application threads are made by the scheduler. But decisions at the router are made locally. We believe user satisfaction as the QoS metric will equally apply for decision making at the application layer and the network layer. Global optimization of user satisfaction is done by the scheduler during allocation of NoC resources to the threads. But this model is extended to router level decision making to have the local influence affect the user satisfaction. We now address the following questions - How can the resources for extra bandwidth in the saturation region for communication flow be reallocated to competing communication flows? How does this translate into VC allocation policy at the router?

6.2 User satisfaction metric versus Quality of service in NoC

NoC based circuit design maps global design goals, of the whole system on chip, into local design choices at local domains. Domains may be a single logic block, a group of logic blocks, or more complex CPU, FPGA. NoC also raises the level of communication abstraction to transaction level enabling local domains communicate by passing messages [DT01]. Figure 6.2 shows a 9 node NoC mesh communication system organized as a 3 X 3 grid.

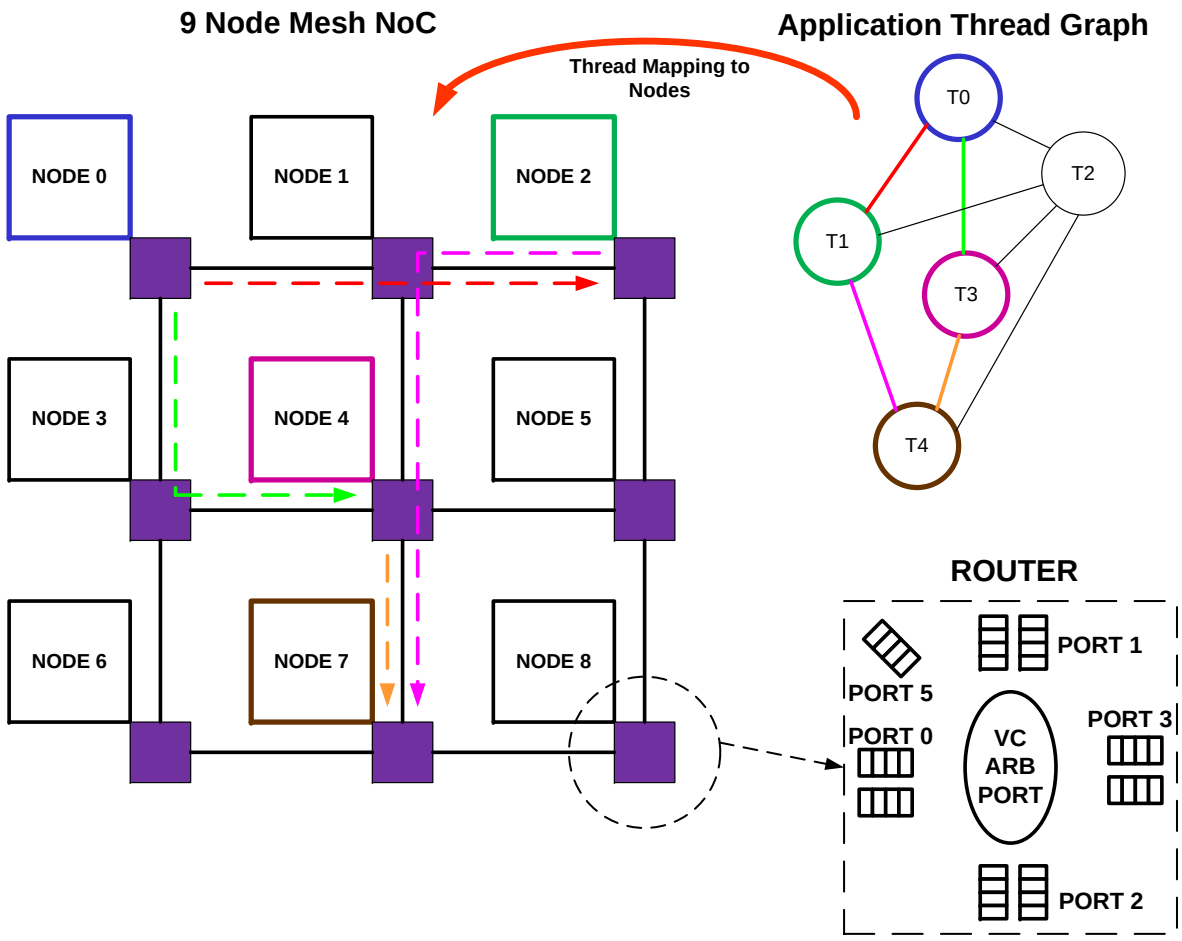

Figure 6.2: NoC mesh - Application mapping

Each node in Figure 6.2 is a local domain which can be independently designed in the SoC. Communication between nodes is facilitated by routers which can buffer and exchange messages. The major elements of NoC router are first in first out (FIFO) queue buffers, virtual channels [Dal92], arbitration protocols and output port buffers. A magnified view of the router on the right side of Figure 6.2 depicts the various internal elements. The router in this example has two input virtual channels per input port which are shown in Figure 6.2. Output ports are not shown for clarity. Each FIFO queue buffer has a depth of four. The simplified control logic consists of three stages - virtual channel allocation (VC), arbitration for output ports (ARB) and port allocation (PORT). Multiple routers lie on the critical path for SoCs using on chip

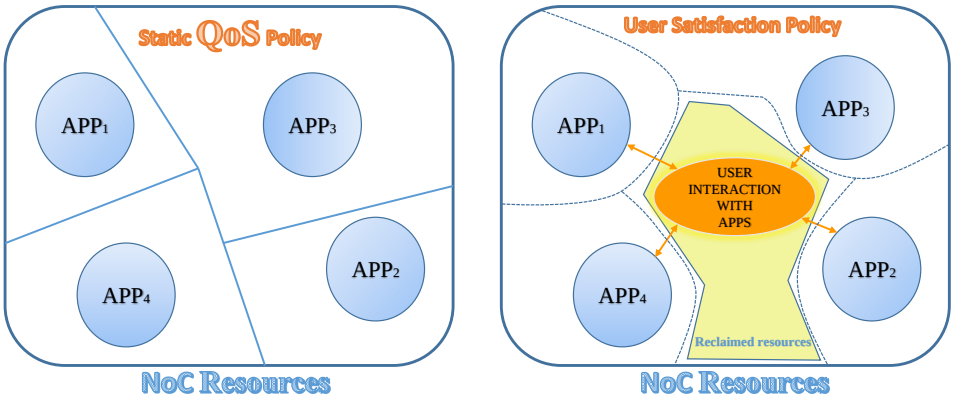

Figure 6.3: User satisfaction and QoS policies for NoC

communication networks.

Design policies for various stages in the control logic and allocation policies for buffers have been explored in several works reviewed in Section 2.2. Buffers are major energy hog in the on chip communication infrastructure. Maximizing utilization of buffers per unit energy spent is of utmost importance to achieve higher throughput of the SoC designs.

Static QoS policy for applications restricts virtual channel allocation flexibility and hence, under-utilizes NoC buffers in two major ways. First, by statically allocating VCs and associated buffers at application level routers are deprived of the flexibility to reallocate free buffers dynamically. Second, the static QoS policies do not reflect the user behavior in embedded systems using NoC paradigm. As compared in Figure 6.3, static QoS allocation policies made at the application layer are difficult to permeate to the NoC layer due to the lack of appropriate model which can be used at both the layers. Existing QoS policy at application level do not provide any control knobs for the NoC layer to improve performance. In fact, each inter-thread communication edge is viewed as an atomic step at this layer. Figure 6.3 shows a possible resource allocation among four applications, APP_1, APP_2, APP_3, APP_4, in the NoC. The boundaries in Figure 6.3 indicate a proportion of the total resources allocated to each application. The demarcation of NoC resources for static QoS policy in Figure 6.3 is rigid and cannot be changed dynamically to adapt to runtime variations in application resource requirements. In our methodology [PT13], the user interaction with the application decides the allocation of

resources at the NoC level. The resource allocation boundaries of applications are flexible due to better matching of resources to dynamic application needs. This is shown by the reclamation of free resources in Figure 6.3. User satisfaction is modeled using the *sigmoid* function (Figure 4.3) in our work. VC allocation decisions are made at routers for communication flows which maximizes the user satisfaction. The sigmoid function is a non-linear saturating function which models the human psycho visual expectation from applications at the application layer. For example, it is known that increasing video playback frames per second (fps) allows for improved viewing of smooth motion under certain conditions. Allocating resources to a video player application to increase the playback frame rate from 24 fps to 35 fps may improve the user satisfaction by a greater amount as opposed to the same resources being used to playback from 50 fps to 55 fps. In the latter case, the human eye is less sensitive to observable smooth motion effects as compared to former case where the playback fps increased from 24 fps to 35 fps. We model this effect using the sigmoid function which saturates when resource allocations exceed a threshold indicated by the knee of the curve in Figure 4.3.

6.3 User satisfaction model for Router

The human psycho visual system responds differently to information received by various senses such as vision and hearing. The end users satisfaction from an embedded system innately depends on the human psycho visual system. Different QoS requirements have to be supported by the NoC depending on application characteristics, implementation, computation needs and importantly the human psycho visual system response to the applications. For example, a video playing at lower frame rate would be more of an annoyance compared to a web page loading slowly. Similarly, audio and text interface based applications have different user satisfaction. Though these are subjective and vary between individual users, modeling them for NoC layer resource optimization is the goal of this section.

The sigmoid model for user satisfaction (UserSat) is mathematically expressed by Equation 6.1.

$$UserSat(t) = \frac{1}{1 + \alpha\ e^{-\beta t}} \tag{6.1}$$

The parameters α and β are used to define the characteristics of the sigmoid curve in Figure 4.3. Parameter β is referred to as the *slope* of the linear region and controls the sensitivity of user satisfaction to the change in resource allocation. The saturation regions at the upper and lower knee of the sigmoid are controlled by the parameter α. The variable t is the physical parameter of interest such as throughput at the application layer, bandwidth of flits at the network layer etc. which influence the user satisfaction. It is constrained by the resource allocation policies.

To develop the objective function for user satisfaction at each router R_i, we develop the constraints on the physical resources which have to be satisfied under all resource allocation cases. Table 6.1 lists the definitions of all the symbols used in the following derivations.

Table 6.1: Notations for optimization of User satisfaction

Symbol	**Definition**
N_i	Node i in NoC mesh
R_i	Router i of node i
L_{ij}	Unidirectional Link from router R_i to R_j
$A_i(T, E)$	i th Application thread graph T - Set of threads $\{T_1^i, T_2^i, \ldots, T_k^i\}$ E - Set of edges between thread $\{e_{mn}^i = 1 \iff T_m^i communicateswith T_n^i\}$
$VC_j^{i,p}$	Virtual channel set of R_i at port p, j = 1,2,...,MAX_VC
sim_cycle_t	Simulation clock cycle time t
$f_{i,m,n}^{sim\_cycle}$	Communication flow from T_m^i to T_n^i at time sim_cycle_t

Condition 6.2 constrains the total number of VCs allocated to all flows at each of the MAX_PORTS number of ports to be less than maximum number of VCs (MAX_VC).

$$\sum_{p=1}^{MAX\_PORTS} \sum_{j=1}^{MAX\_VC} VC_j^{i,p}(f_{r,m,n}^{sim\_cycle_t}) \leq 4 * MAX\_VC \tag{6.2}$$

$VC_j^{i,p}(f_{r,m,n}^{sim\_cycle_t}) = 1$ if jth VC at port p is allocated to flow $f_{i,m,n}^{sim\_cycle_t}$.

Condition 6.3 is the second constraint on the arbitration for the output buffer at router R_i.

$$\sum_{i=1}^{MAX\_PORTS} ARB_i^{sim\_cycle_v}(f_{r,m,n}^{sim\_cycle_t}) \leq MAX\_PORTS \tag{6.3}$$

This condition expresses the fact that only one flow's packet can reach the output buffer of any port at a given simulation time v.

The bandwidth of flow, $f_{r,m,n}^{sim\_cycle_t}$, over time duration Δt, at router R_i can be calculated subject to Conditions 6.2, 6.3 as follows.

$$BW(f_{r,m,n}^{sim\_cycle_t}) = \sum_{v=1}^{\Delta t} \sum_{i=1}^{MAX\_PORTS} [\frac{VC_j^{i,p}(f_{r,m,n}^{sim\_cycle_t}) * ARB_i^{sim\_cycle_v}(f_{r,m,n}^{sim\_cycle_t})}{\Delta t}] \tag{6.4}$$

Note that Equation 6.4 applies for any policy chosen for VC allocation, arbitration and routing. For example, the flow can exit through any of the router's port due to adaptive routing. Equation 6.4 adds up the bandwidth of the flow for all the ports. The throttling factor (TRF) of flow, $f_{r,m,n}^{sim\_cycle_t}$, due to other flows competing for resources at router R_i is defined as,

$$TRF^i(f_{r,m,n}^{sim\_cycle_t}) = \frac{BW(f_{r,m,n}^{sim\_cycle_t})}{BW\ of\ all\ flows\ at\ R_i} \tag{6.5}$$

Substituting Equation 6.5 in Equation 6.1, we derive the effect of resource allocation at router R_i on the user satisfaction of communication flow from thread, T_m^r, to thread, T_n^r, at simulation time t.

$$UserSat(f_{r,m,n}^{sim\_cycle_t}) = \frac{1}{1 + \alpha\ e^{-\beta * TRF^i(f_{r,m,n}^{sim\_cycle_t})}} \tag{6.6}$$

The communication edge between T_m^r and T_n^r is mapped to a subset of routers, P. Data packets from T_m^r to T_n^r hop along the routers in P. The overall effect of resource allocation at all the routers on the communication flow's path in the network is expressed as follows.

$$UserSat(f_{r,m,n}^{sim\_cycle_t}) = \frac{1}{1 + \alpha\ e^{-\beta * \prod_{p \in P} TRF^p(f_{r,m,n}^{sim\_cycle_t})}} \tag{6.7}$$

We will use Equation 6.6 in evaluating virtual channel allocation for competing communication flows and also arbitration for output port once VC allocation is complete.

6.4 User satisfaction modeling of NoC energy utilization

Energy utilization modeling at the NOC layer is also investigated in this work to quantify the effectiveness of application layer user satisfaction policies on energy utilization of NoC resources. We attempt to model user behavior at distinct energy profiles to influence the resource allocation policies using the sigmoid model described in Section 4.4. Existing research efforts [HM04b, HM05, MCM+05] have attempted to model the energy consumption by estimating the number of hops between source and destination nodes in the NOC for each packet. Research related to quality of service [HKKB13] for applications ignores energy due to the fact that it is an indirect parameter in system design and often not specified in the expected design characteristics of the application such as frame rate, throughput. Such models do not consider the problem of resource allocation to various flows of communicating threads to optimize energy consumption predicated on the user behavior at any given instant. Further, there is a disconnect between QoS based routing and its impact on energy consumption at NoC layer. We address this issue in the following discussion.

The behavior of user is different on the time and energy axes [PT14] in the application design space. However, the space of user behavior is infinite considering the vast number of users who have different expectations from the same embedded system. Modeling user behavior for different system energy profiles is akin to finding eigen vectors to efficiently express complex functions as projections along eigen axes. Consequently, user satisfaction policies for energy optimization can be projected from the application design space to the NoC abstraction. The time axis refers to attributes such as latency & response time of the application. A text based chat application is expected to respond quickly to user actions and increased response times will result in diminished user satisfaction. Similarly, latency of decoding video frames is expected to be within the supported frame rate to avoid choppiness in the video which may also result in diminished user satisfaction. Such problems fit into the sigmoid model proposed in Section 4.4. Hence, policies at the NoC affect the user satisfaction [PT14] in a direct manner during short term resource allocations to improve application performance.

In contrast, the effects of energy optimization policies are not apparent to the user until the fixed energy source, E_{source}, decreases to a critical level, $E_{critical}$. The critical level may vary with the user. User behavior may change if current energy reserve, $E_{current}$, falls below $E_{critical}$. The user may choose to ignore the critical nature of the situation by continuing existing status quo in the embedded system. For example, the user may continue to run all active applications, A, some of which are energy demanding such as watching a HD video until system shutdown. However, in other cases, the user may choose to change the status of the embedded system by implicitly trying to extend the energy source for longer duration. For example, the user may choose to quit a set of energy consuming applications which are in the *D*o *N*ot *P*refer list, $A_{DNP} \subset A$, such as HD video movie decoder. The remaining set of applications are in the *C*urrently *P*referred *A*pplications list, $A_{CPA} \subset A$, such as e-mail client, web surfing, which use the remaining energy, $E_{critical}$. The user satisfaction policy changes for A_{CPA} when compared to A_{DNP}. Hence, satisfying the constraints of applications in the set A_{CPA} maximizes the user satisfaction in the region, $E_{current} \leq E_{critical}$. In this discussion, we have identified two distinct regions in the user satisfaction versus energy axis to be the energy profiles for the NoC based embedded systems. The regions are illustrated in the user satisfaction model in Figure 6.4. Resource allocation to applications in A_{CPA} result in more user satisfaction compared to applications in A_{DNP} which anyway will drain the energy source at a faster rate in the region, $E_{current} \leq E_{critical}$. However, the secondary goal of our methodology is to reduce the spread between the two curves in Figure 6.4. This basically translates to the NoC layer attempting to run applications in A_{DNP} while satisfying the constraints of A_{CPA}. Note that we implicitly assume that the applications in A_{CPA} can last longer in the region $E_{current} \leq E_{critical}$ when compared to all applications in A. This is justified since text based applications such as chat, quick updates to social media platforms etc. are less compute intensive and hence, consume lower energy. Further, the user of a mobile system is aware of the existence of a minimal set A_{CPA} from his/her own experience. The objective of the NoC optimization is to allocate resources to maximize the user satisfaction under these conditions.

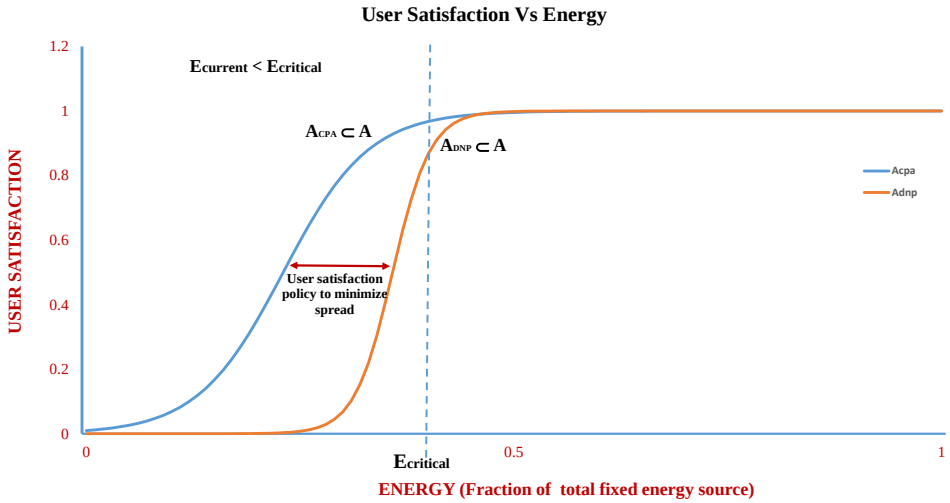

Figure 6.4: Sigmoid function for Energy modeling

6.5 NoC energy utilization analysis

The application model consists of a set of N applications, $A = \{ A_i \}, i = 0, 1, \ldots, N-1$. Each application, A_i, is implemented as a group of C_i communicating threads, $T = \left\{ T_i^j \right\}, j = 0, 1, \ldots, C_i - 1$. We estimate the energy consumed over the life of an application by aggregating the energy consumed during the buffering overhead of packets at each router node and the links traversed in the path from source to destination over all pairs of communicating threads. For each pair of communicating threads of an application A_m, T_m^p and T_m^q ; $p \neq q$, that are mapped onto the nodes of NoC, the aggregate energy consumption is calculated as below in Equation 6.8.

$$E_{p,q}^m = N_{cycles} * E_{buffer} + (N_{hops} - 1) * E_{link} \tag{6.8}$$

E_{buffer} and E_{link} are the energy consumption of a single buffer and a single link in the NoC router and interconnect fabric respectively. N_{hops} is the number of hops each packet travels from source thread, T_m^p, to destination thread, T_m^q. N_{cycles} is the total number of cycles (in the simulation) each packet takes to complete N_{hops}. N_{cycles} includes delays introduced by congestion, VC allocation and competition for output ports. In the case of congestion free

routing, the nodes and routers along the path of a packet form an extended pipeline to deliver the packet to its final destination. Hence, N_{cycles} can be split into two components – stall cycles, N_{stall} and hop cycles, N_{hops} as shown in Equation 6.9. N_{stall} accounts for the total number of simulation cycles a packet may wait at intermediate routers before arrival at final destination.

$$N_{cycles} = N_{stall} + N_{hops} \tag{6.9}$$

Substituting Equation 6.9 in Equation 6.8 and rearranging terms, the aggregate energy for a single packet can also be split into two components as in Equation 6.12.

$$E^m_{p,q} = (N_{stall} + N_{hops}) * E_{buffer} + (N_{hops} - 1) * E_{link} \tag{6.10}$$

$$E^m_{p,q} = N_{stall} * E_{buffer} + (N_{hops} * E_{buffer} + (N_{hops} - 1) * E_{link}) \tag{6.11}$$

$$\begin{aligned} E^m_{p,q} \quad = \quad & E^{m_{NOC}}_{p,q} + E^{m_{SCHED}}_{p,q}, \\ & where,\ E^{m_{NOC}}_{p,q} = N_{stall} * E_{buffer}\ and \\ & E^{m_{SCHED}}_{p,q} = N_{hops} * E_{buffer} + (N_{hops} - 1) * E_{link} \end{aligned} \tag{6.12}$$

$E^{m_{NOC}}_{p,q}$ is the component of total energy which is incurred due to resource allocation policies in the NoC layer. $E^{m_{SCHED}}_{p,q}$ is affected by scheduling and placement policies at the application thread layer. In this section, user satisfaction policy is developed to optimize $E^{m_{NOC}}_{p,q}$ at the NoC layer and hence, maximize user satisfaction by tuning VC allocation and arbitration to prioritize packets of applications in the critical set in the region $E_{current} \leq E_{critical}$ of Figure 6.4.

The total energy consumption of the application, A_m, over all the flows from all pairs of thread communication in the NoC is given by Equation 6.13.

$$E^m = \sum_{F_m} \sum_{p,q} \left(E^{m_{NOC}}_{p,q} + E^{m_{SCHED}}_{p,q} \right) \tag{6.13}$$

F_m is the set of all communication data flows between threads in the application A_m. Similarly, the total energy consumption of A_{CPA} and A_{DNP} is the summation over each individual application in that set and is given by Equations 6.14 and 6.15.

$$E^{A_{CPA}} = \sum_{A_m \in A_{CPA}} \sum_{F_m} \sum_{p,q} \left(E_{p,q}^{m_{NOC}} + E_{p,q}^{m_{SCHED}} \right) \tag{6.14}$$

$$E^{A_{DNP}} = \sum_{A_m \in A_{DNP}} \sum_{F_m} \sum_{p,q} \left(E_{p,q}^{m_{NOC}} + E_{p,q}^{m_{SCHED}} \right) \tag{6.15}$$

By fine tuning resource allocation at the NoC layer, under-utilized buffers from applications in A_{CPA} can be reallocated in our scheme to benefit applications in A_{DNP} under the conditions that user satisfaction is not affected for applications in A_{CPA}. Hence, energy in the region, $E_{current} \leq E_{critical}$, is not wasted on idle resources. This increases the user satisfaction for applications in both sets A_{CPA} and A_{DNP}. We use Equations 6.2, 6.3 and 6.7 to allocate and share VCs for applications in A_{CPA} and A_{DNP}.

6.6 Virtual Channel Allocation Heuristic

Virtual channel allocation policy implements shared VC policy between competing flows. Sharing of VC dynamically increases the number of VCs available at each port. However, it does decrease the depth of the FIFO queue for each flow. The VC allocation logic is modified to reflect the sharing of virtual channels as shown in Figure 6.5. If a communication flow is near the saturation region, the VC occupied by that flow can be shared with a competing flow provided free buffer space is available. This technique maximizes the total user satisfaction of both the flows.

A greedy heuristic which maximizes the user satisfaction for the flows is used to allocate VC and arbitrate for output port at each router.

1. For each port, find the set of flows, S, competing for VCs. If cardinality of set $S \leq$ number of VCs, there are enough VCs for the set S. Else go to Step 2.

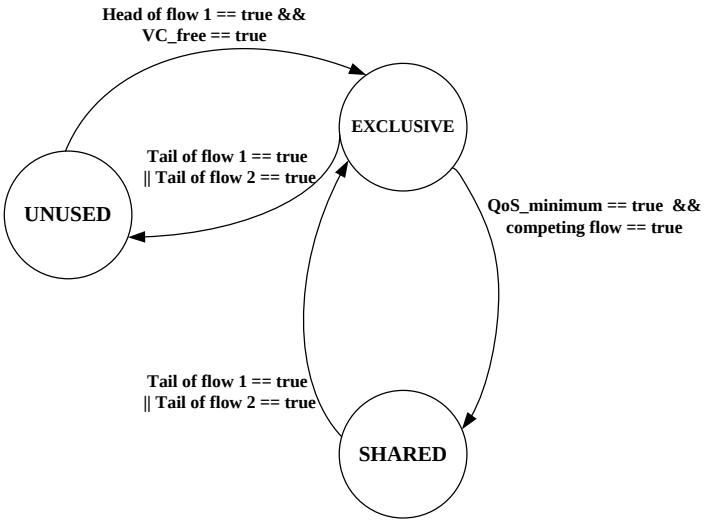

Figure 6.5: VC States for shared communication flows

2. Use Equation 6.6 to determine the user satisfaction at the router for each flow of set S. Choose the subset of flows from set S which maximizes the sum of user satisfaction subject to constraints in Equation 6.2.

3. Allocate VCs to the subset of flows chosen from Step 2.

4. Check if user satisfaction of the subset of flows chosen from Step 2 due to resource allocation in Step 3 is in the saturation region of the sigmoid model.

5. If Step 4 == true, modify the state of VCs for this flow to be shared with a competing flow that maximizes uses satisfaction. This step dynamically increases the number of VCs by one. Else go to Step 1.

6.7 Experiments & Results

We use four simulator configurations to evaluate the effectiveness of the proposed scheme on a 4X4 NoC network. Routers are modeled as ideal resources for the first configuration, *Ideal*. The throttling factors are set to unity for the routers in the flow path in the network during mapping of threads to NoC nodes. The next three configurations model a non-ideal router. A

base case configuration, *Base*, of the simulator uses round robin allocation for virtual channels and output port arbitration on the downstream router. The application threads do not have any QoS guarantees. The third configuration, *Qos*, guarantees QoS for the video application threads but other application threads do not have any QoS guarantees. The final configuration, *usersat*, implements the ideas enunciated in this work. Sharing of VC is enabled for competing flows when QoS guarantee is reached (saturation region). The results in Figure 6.6 and Figure 6.7 show the plot of average user satisfaction over all threads for the MPEG-2 and MP3 applications for the four configurations. Figure 6.6 and Figure 6.7 shows the bar graphs for 4, 8, 16 and 32 VC configuration for each port of the router. Sharing of VCs for the *usersat* configuration can at most double the VCs to maximize user satisfaction.

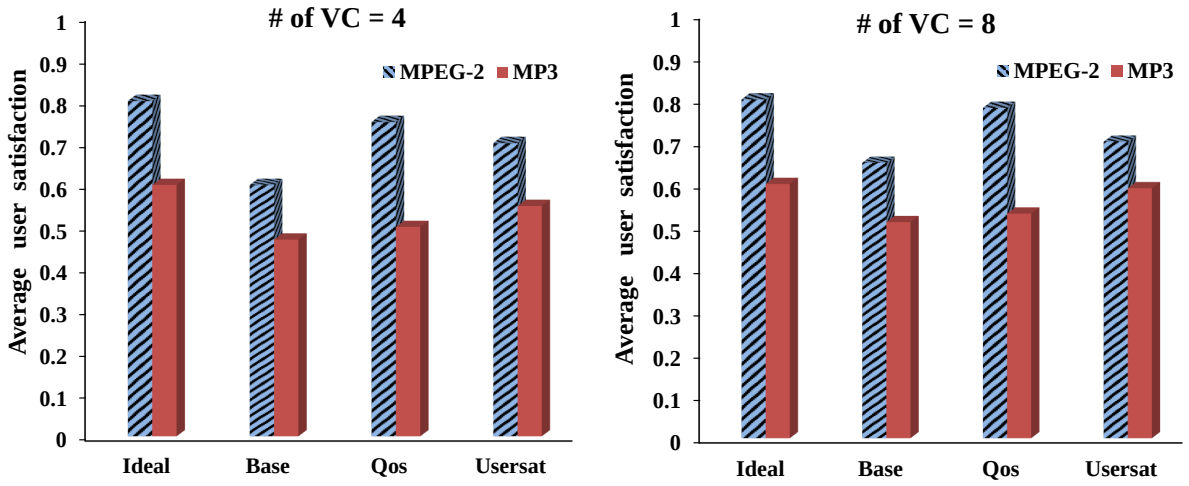

Figure 6.6: Average user satisfaction for VC = 4,8

The results show that user satisfaction is maximum for the *Ideal* case since there is no attenuation of flow bandwidth by routers in the communication path in the NoC network. There is significant degradation in user satisfaction for the *Base* case due to the fact that all threads from both applications compete equally for the router resources in round robin fashion. In the *Qos* case, the video application threads have guaranteed user satisfaction of 0.7. This is reflected in the figures above where the video application user satisfaction is more than 0.7. However, extra buffer resources allocated to the VC of video communication flow do not benefit any other competing flows. The competing MP3 flow suffers in user satisfaction and

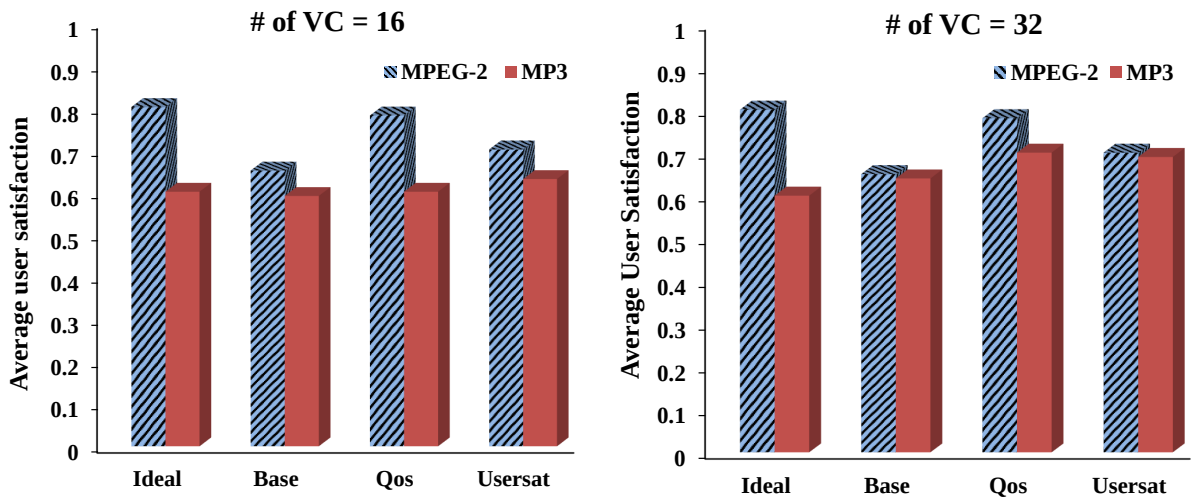

Figure 6.7: Average user satisfaction for VC = 16,32

there not much improvement as compared to the *Base* case. The extra user satisfaction past 0.7 for the video application is in the saturation region. It does not result in significant user satisfaction for the buffer or VC resources allocated. However, the *usersat* case proposed in this work reallocated VC resources to MP3 flow by sharing VC with the MPEG-2 flow. The user satisfaction of MP3 flow improves by 10% when VC = 4 and 11% when VC = 8 while still guaranteeing a user satisfaction of 0.7 for MPEG-2 flow. When the number of VCs is increased the user satisfaction improves overall. This is because more router resources are available for flows and the effects are clearly demonstrated in Figure 6.7. The user satisfaction of the MPEG-2 and MP3 applications in the *usersat* and *Qos* simulations are nearly the same in Figure 6.7. Due to less contention for VC resources, the data flows from multiple applications do not suffer as much degradation in bandwidth as in Figure 6.6. Hence, user satisfaction plays important role in resource allocation when system resources are in demand from various applications and under-utilization of resources are possible in the saturation regions of applications. It should be noted that the 10% improvement is the aggregate of the user satisfaction due to resource allocation at multiple routers in the communications flow path. The results definitely demonstrate the effectiveness to harvesting extra resources in the saturation region of Figure 4.3 for maximizing user satisfaction.

The goal of the next experiment is to evaluate the effectiveness of user satisfaction model for energy optimization of NoC resources as discussed in Section 6.4. The energy source is limited for the applications in this simulation configuration. $E_{critical}$ is defined arbitrarily as 20% of the total energy source. We consider two scenarios to validate the claims in Section 6.4. In the first scenario, $USECASE_1$, all application threads compete for resources using their user satisfaction metric without regard to $E_{critical}$. We expect the energy source to exhaust faster due to the fact that applications can tie up resources in the region, $E_{current} \leq E_{critical}$ (see Section 6.4). In the second scenario, $USECASE_2$, a set of applications are designated as the currently preferred set, A_{CPA} and these are chosen to be less compute intensive compared to the $A_{DNP} = \{MPEG-2, MP3\}$ applications. These are synthetic applications, $A_{CPA} = \{TEXT1, TEXT2\}$, which mimic texting application. Hence, threads of these applications send few packets per communication and occupy comparatively less resources. Five different simulation runs, RUN_1 through RUN_5, invoke applications at different frequencies and order of execution. The scenario which can prolong the energy source and simultaneously, execute applications in both the set A_{CPA} and A_{DNP} is expected to maximize user satisfaction metric. The energy consumption for each of $TEXT1$, $TEXT2$, $MPEG-2$, $MP3$ application is estimated using Equation 6.13 for all possible placement of threads in the 4X4 NoC. The average of the energy, E^{av}_{app}, for all possible placement of an application is used as reference to estimate the total fixed energy source value, E_{source}. The values for E_{router} and E_{link} are derived from the NoC router model proposed in [BWM+09]. The router model of [BWM+09] is similar to the architecture used in this work. In our case, we arbitrarily choose $E_{source} = 10 * (E^{av}_{TEXT1} + E^{av}_{TEXT2} + E^{av}_{MPEG-2} + E^{av}_{MP3})$. This implies that each of the application can be invoked 10 times, assuming random placement, before the energy source gets exhausted. In each of the scenarios, the user invokes the different applications the same number of times but at randomly chosen simulation cycles. In $USECASE_1$, the model discussed in Section 6.3 is used to allocate VC and calculate user satisfaction metric. In $USECASE_2$, the energy optimization model discussed in Section 6.4 is used to allocate VC and calculate user satisfaction metric such that user satisfaction for A_{CPA} is maximum.

The results of energy optimization for $USECASE_1$ and $USECASE_2$ are summarized in the graphs of Figure 6.8 through Figure 6.12.

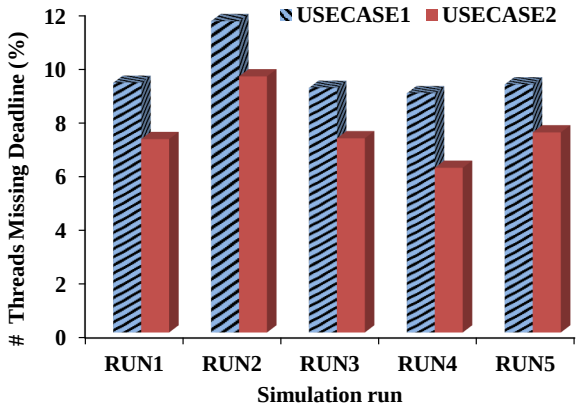

Figure 6.8: Number of threads missing deadlines

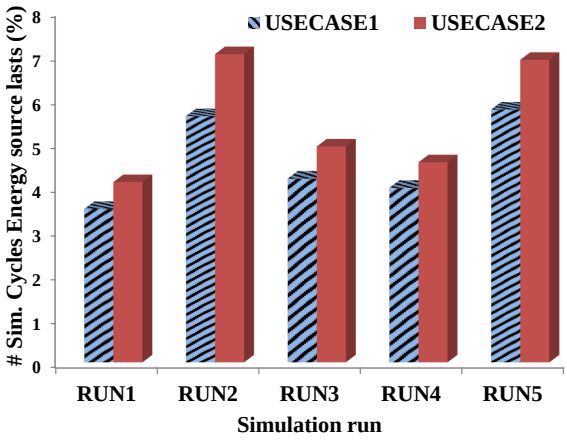

Figure 6.9: Duration of Energy source

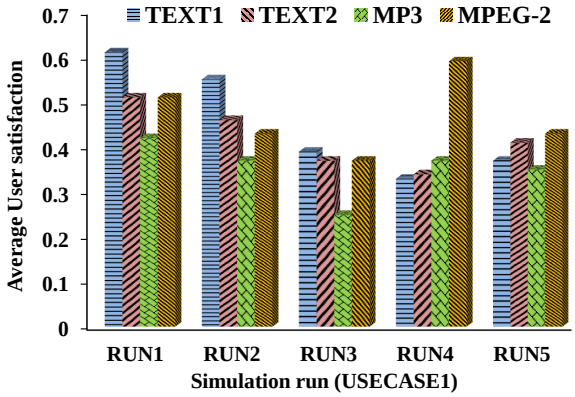

Figure 6.10: Average User satisfaction - $USECASE_1$

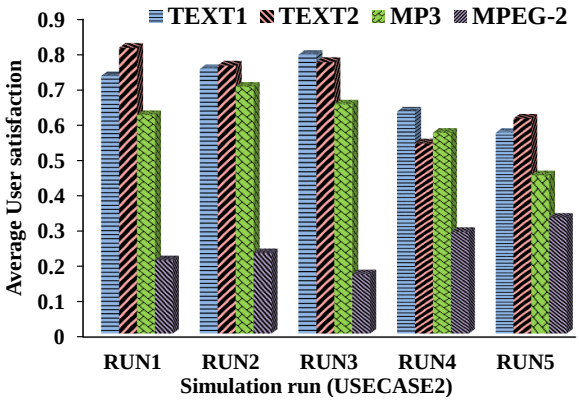

Figure 6.11: Average User satisfaction - $USECASE_2$

The number of threads from all applications which miss the deadline due to the fixed energy source getting exhausted is shown in Figure 6.8. *x-axis* shows the various simulation runs and *y-axis* shows the fraction of threads which miss the deadline for $USECASE_1$ and $USECASE_2$ compared to the total number of threads executed with unlimited energy source. The user satisfaction model discussed in Section 6.4 results in lower number of threads missing the deadline for $USECASE_2$ due to the fact that buffers in VC can be efficiently allocated to

applications in the set, A_{DNP}, when less compute and communication intensive applications are active in the region, $E_{critical}$. Figure 6.9 also validates the previous conclusion further by plotting the fraction of simulation cycles the energy source can last for $USECASE_1$ and $USECASE_2$ compared with the number of simulation cycles with unlimited energy source.

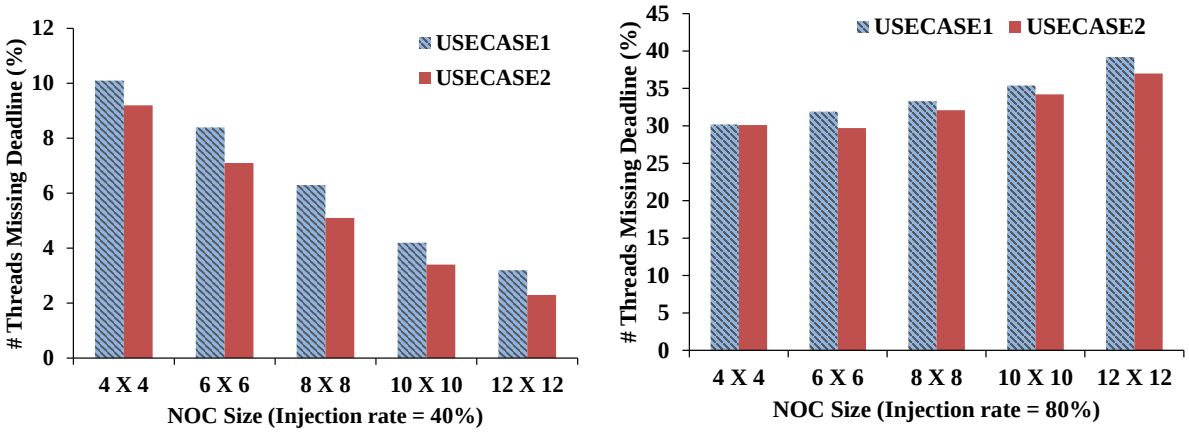

Figure 6.12: # Threads Missing Deadline in $E_{critical}$ region for two injection rates

The percentage of threads missing deadline for injection rate (40%) which is in the neighborhood of the saturation of our NOC simulator acceptance rate is shown in Figure 6.12. As the injection rates hover around the neighborhood of the saturation of NOC acceptance rate, we observe that the user satisfaction model ($USECASE_2$ in Figure 6.12) pushes more threads to completion (on an average of 1%) compared to $USECASE_1$. This is a desirable characteristic of our model which comes to light during higher resource contention when NOC saturation starts to set in. However, the limitations of user satisfaction model in the $E_{critical}$ region is apparent for higher injection rates and we have presented the results in Figure 6.12 for 80% network injection rate. Our model barely improves the thread completion rate due to heavy contention for resources after network saturation and roughly half of the threads spawned miss the deadline in the $E_{critical}$ region. This has a negative impact on user satisfaction as discussed later in this section (Figure 6.13). The methodology in Section 6.4 extends the energy source for an average of 18.8% over $USECASE1$. The average user satisfaction in the $E_{critical}$ region is plotted in Figure 6.10 and Figure 6.11 for applications of $USECASE_1$ and $USECASE_2$ respectively for various simulation runs. The extension of energy source improves the user sat-

isfaction over all the simulation runs as seen in Figure 6.11 for applications in the set, A_{CPA}. It also improves the user satisfaction of the $MP3$ application which is in the A_{DNP}. This because there are enough VCs which can be shared with MP3 communication flows in the $E_{critical}$ region. However, in RUN_4 and RUN_5, the user invokes the $MPEG-2$ application which has a negative impact on the user satisfaction of $TEXT1$ and $TEXT2$ applications. We conclude that in the region, $E_{critical}$, the choice of applications affects resource allocation and hence, energy consumption from the perspective of user satisfaction metric. Our model is successful in allocating resources to applications in the A_{DNP} set which definitely improves overall user satisfaction without too much impact on the energy source.

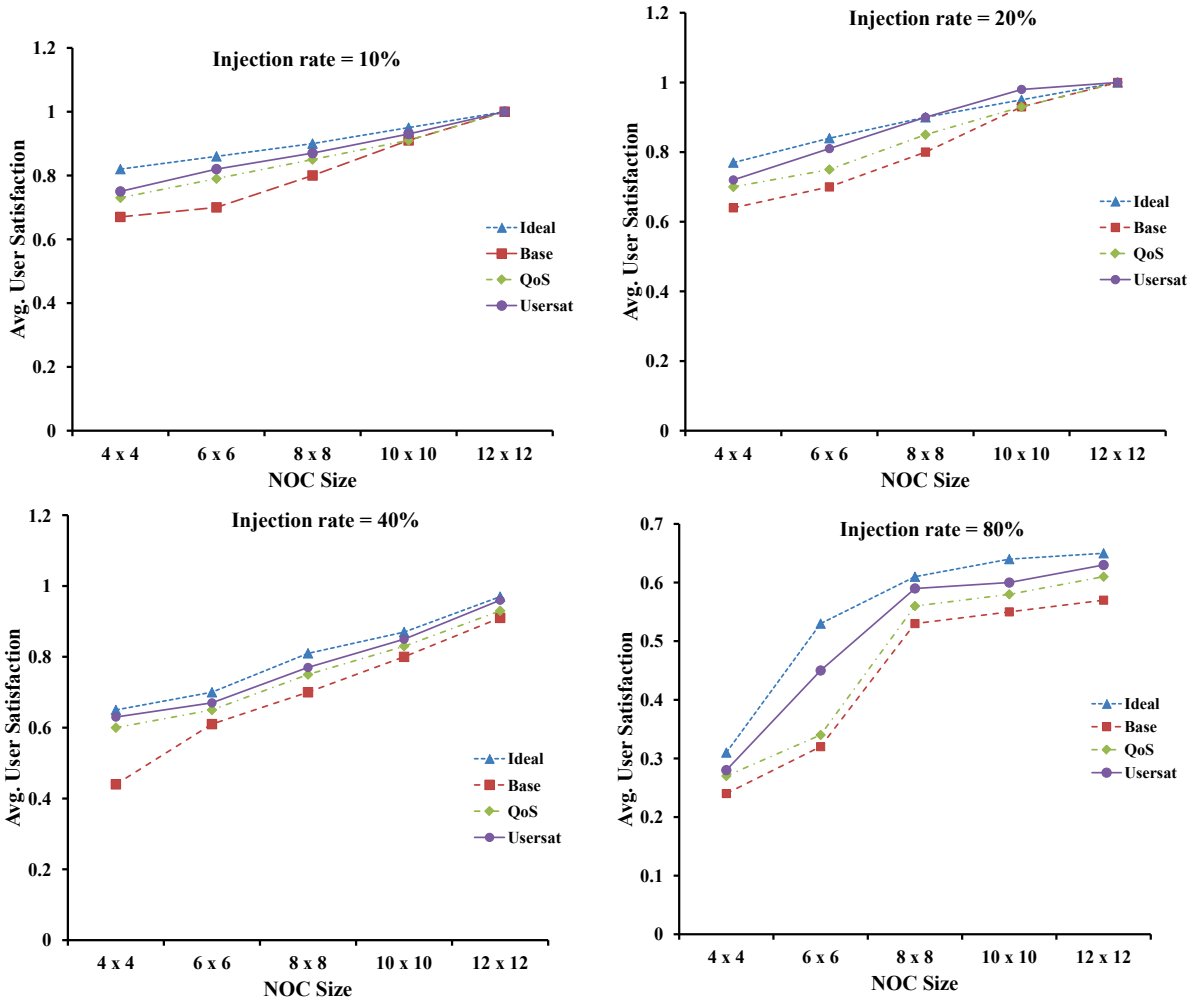

Figure 6.13: User satisfaction trends for scaled NOC sizes

The trends in user satisfaction metric for the four VC allocation strategies are plotted in Figure 6.13 by varying NOC sizes. Average user satisfaction is determined for injection rates ranging from 10% to 80% for each NOC size. User satisfaction (*usersat*) based VC allocation tends to outperform *Qos* and *Base* schemes for lower resources (small NOC sizes) and higher congestion (higher injection rates). For small NOC size (4X4) and/or higher congestion (80%), *usersat* is effective in reallocating resources in the saturation region. This results in better experience for the user when the system is under tight constraints. Under medium injection rates (40% and 60%), the four schemes perform equally well due to the fact that perturbation in resource allocation has marginal effect of user satisfaction. A similar effect is noticeable for medium sized NOC under light injection rates. *usersat* scheme is at par with other schemes due to the added flexibility in scheduling threads on cores due to higher resource availability for larger NOC compounded by low injection rates. Under higher injection rates (80%), the user satisfaction contours separate farther away and overall, the *usersat* scheme performs better for various NOC sizes. It can be observed that average user satisfaction is lower when the injection rate is 80%. The NOC accepted traffic saturates at about 38% of the injected traffic. The packet delay increases even with resource reallocation and hence, results in negative impact on user satisfaction. Given plenty of resources (12X12 NOC) and lighter injection rates (10%), all schemes perform equally well and the contours coincide for large NOC size seen in Figure 6.13. The plots in Figure 6.13 qualifies the domain of *usersat* scheme's effectiveness in terms of NOC size and injection rate.

CHAPTER 7. DYNAMIC BINARY TRANSLATOR FOR PolyNoC FRAMEWORK

Dynamic binary translator (DBT) implementations manage dynamic code transformations [BDB00, LCM+05]. A section of code is transformed for optimizing one or more application parameters while maintaining the program correctness. Mobile OS frameworks use DBT for isolating application execution for security, dynamic compilation of code and runtime optimization of code sections. Android OS uses Dalvik [Dal] virtual machine for isolating application execution context from other applications and uses Dalvik just in time compilation to optimize code while the application is executing. PolyNoC should be capable of simulating applications that use JIT like layer for runtime optimization. Integrating DBT framework in PolyNoC provides complete simulation ecosystem for design space exploration of heterogeneous multicore architectures running Android/Linux like OS. This chapter provides details about integration of our Tightly Coupled Monitoring (TCM) DBT framework in PolyNoC.

The architecture of PolyNoC simulation environment augmented with TCM framework is shown in Figure 7.1. The main difference from the PolyNoC implementation in Chapter 5 is that applications are encapsulated in the TCM DBT runtime environment. In Figure 7.1, the MPEG-2 application executes under the control of the TCM framework. TCM framework takes control of the execution of MPEG-2 application by trapping calls to the polymorphic layer. The MPEG-2 application also has access to internal states of the NoC layer, scheduling and packet management layer. Internal states such as congestion information, flit rate can be used in wrappers, introduced in Section 7.1, to further optimize the performance of the application. Next, we discuss the details of TCM framework and enlighten the benefits of using wrappers to access NoC internal states at runtime.

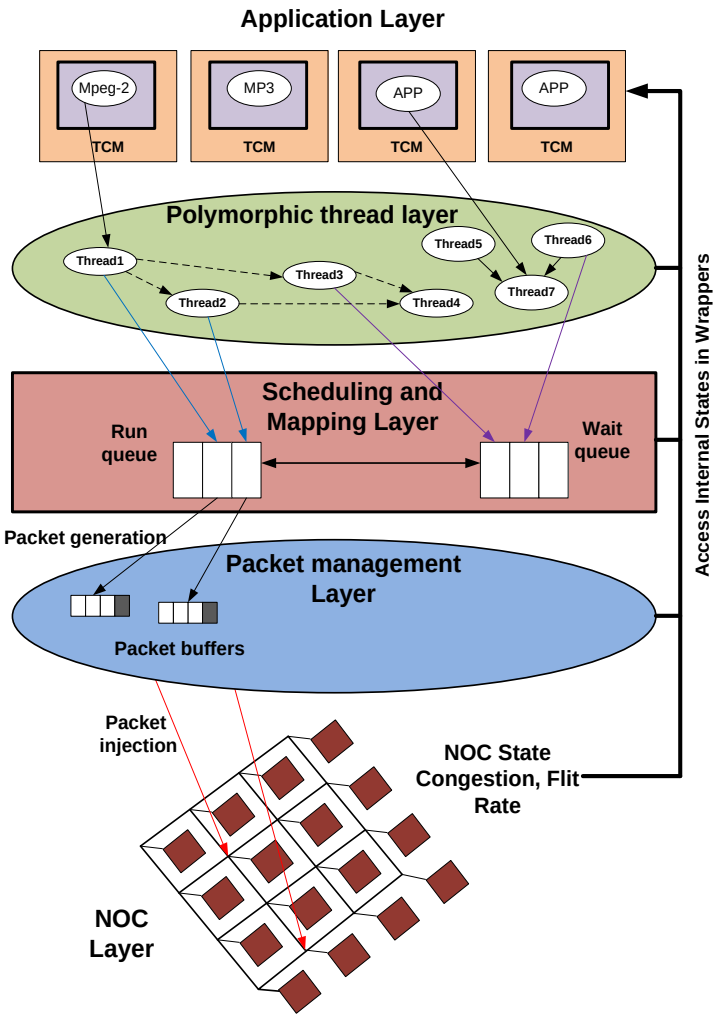

Figure 7.1: PolyNoC with TCM Framework

7.1 Tightly Coupled Monitoring Framework

DBT mechanism is abstracted as a tightly coupled interaction between one or more transformations and the corresponding code object in the TCM framework. Figure 7.2 illustrates the abstract layers through which an application can interact with the *Tightly Coupled Monitoring* (TCM) framework. *Tightly coupled* interaction of Transformation Execution Engine (*TEE*) and Code Object Execution Engine (*COEE*) in the DBT layer is the novelty of our TCM framework. NoC layer events are stable over a small window. Tight coupling ensures timely code optimization which depend on such events. Transformations are executed by TEE. Any input *states* used by the transformations are requested by TEE from the TCM framework

core. *COEE* executes code objects from application code. Code objects modified by TEE is passed on to COEE for execution. Unmodified code objects from the application binary file is directly executed by COEE. Program variables, register state and hardware machine state of *COEE* are updated after execution of code object instructions. This novel abstraction enables transformations to be triggered by run time state values in a small time window. TCM framework core manages data structures for both TEE and COEE.

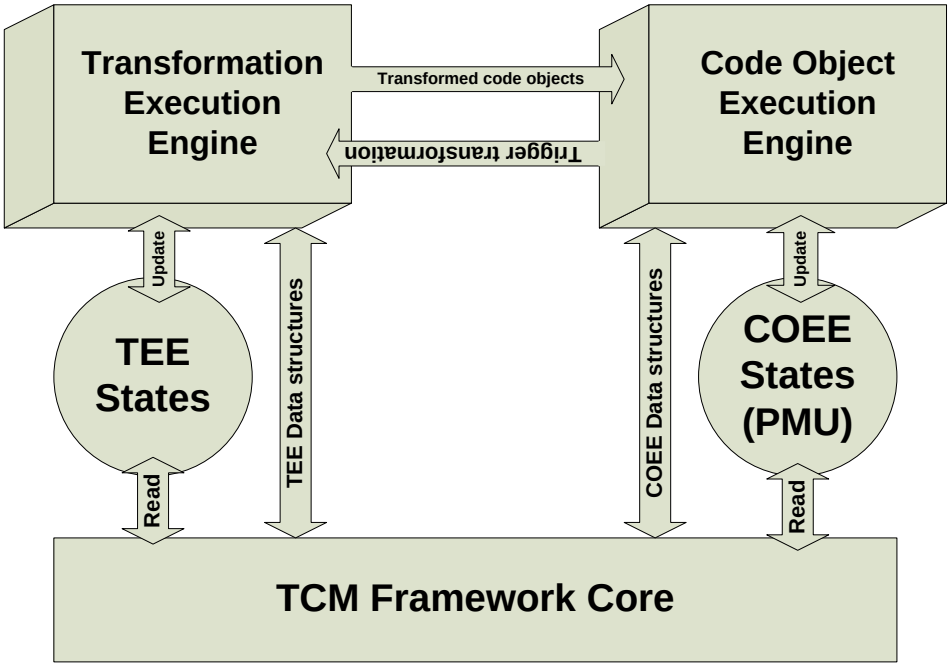

Figure 7.2: Tightly Coupled Monitoring Framework

Application code is interspersed with transformation wrappers to take advantage of the TCM framework. Transformation wrappers are code segments which implement the transformations. We use the term *wrappers* to signify that transformations wrap the code objects and modifies it if needed. Inserting wrappers into application code requires two new keywords to demarcate code segments implementing the transformation. *Wrapper code* section is enclosed between the tags <*wrapper_code*> and </*wrapper_code*>. *Wrapped code* or code object is enclosed between the tags <*wrapped_code*> and </*wrapped_code*>. Application code is interspersed with wrapper code at the granularity of optimization required.

For example, wrapper code can be inserted into the program code at function level, basic block level etc.

Our framework simplifies specification of code optimizations. For example, loop index is statically unknown to the compiler for the x86 code in Figure 7.3 but the index variable is known dynamically before the loop code gets executed. The state of the loop index variable is read by *TEE* and the appropriate wrapper code for optimizing the loop code is executed by *TEE* using the new loop index information. The optimized wrapped code (the *for* loop code in Figure 7.3) is then executed by *COEE*.

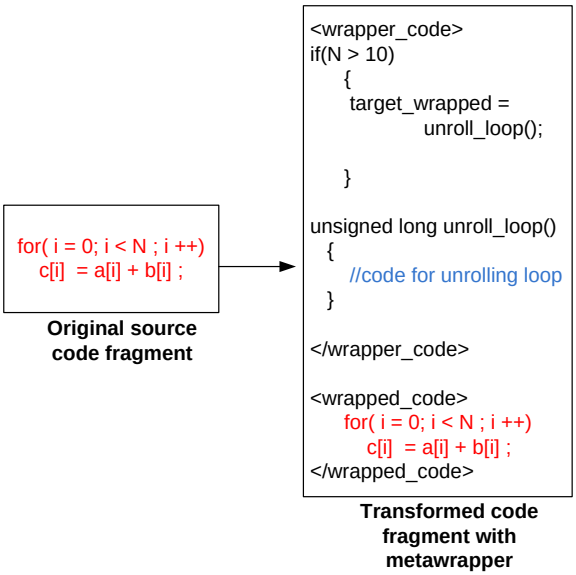

Figure 7.3: Pseudo wrapper code for loop optimization

Dynamic optimizations specified using wrappers in *TCM* framework resemble the program code more naturally. Transformations look like regular functions in *C* language. Note that this model of specifying optimization extends our framework to provide general monitoring functions other than dynamic optimizations. For example, the stack pointer of the application can be monitored in the wrapper to prevent buffer overflow attacks. This framework promotes the idea of writing code for optimization using run time data available in a small time window before the execution of wrapper code. Our framework's tight-coupling enables such optimizations.

7.2 Experiments & Results

To evaluate the effectiveness of wrappers in Android like framework, we have modified the polymorphic MPEG-2 decoder application to work with the TCM framework of Section 7.1. The congestion state of the router associated with the node executing the MPEG-2 application thread is available as a shared variable through the TCM framework. The application thread can access this variable to control the forking of new threads depending on the congestion status of the NoC layer. Overall execution time of MPEG-2 application with wrapper is compared for various network congestion conditions to understand the advantage of using wrapper. The results are summarized in Tables 7.1, 7.2, 7.3 and 7.4. Under low network congestion (Tables 7.1and 7.2), there is not much difference in the execution times of the MPEG-2 application. This is because the wrapper is not activated under low congestion. We expect the wrapper to get activated under high congestion conditions to control the creation of new threads. New threads worsen the congestion more by trying to push packets into the links when NoC layer is already backlogged. We can see from Table 7.3 and 7.4 that the wrapper improves execution time of the MPEG-2 application by 9.25%. This experiment shows that TCM framework is beneficial for PolyNoC simulation environment and similar scenarios in real embedded system development framework such as Android can be modeled with our tool.

Table 7.1: MPEG-2 Decoder Execution time 1 - PolyNoC TCM Framework

Injection rate = 10%		
	Execution time (in simulated cycles)	
Input	**Wrapper**	**No Wrapper**
tennis.mpg	25252	24345
football.mpg	18683	21980
garden.mpg	35623	33115
salesm.mpg	47813	45231

Table 7.2: MPEG-2 Decoder Execution time 2 - PolyNoC TCM Framework

Injection rate = 20%		
	Execution time (in simulated cycles)	
Input	**Wrapper**	**No Wrapper**
tennis.mpg	26212	24023
football.mpg	18603	20139
garden.mpg	34893	34431
salesm.mpg	47433	45145

Table 7.3: MPEG-2 Decoder Execution time 3 - PolyNoC TCM Framework

Injection rate = 40%		
	Execution time (in simulated cycles)	
Input	**Wrapper**	**No Wrapper**
tennis.mpg	28320	30045
football.mpg	22381	29284
garden.mpg	35623	38216
salesm.mpg	53016	60457

Table 7.4: MPEG-2 Decoder Execution time 4 - PolyNoC TCM Framework

Injection rate = 80%		
	Execution time (in simulated cycles)	
Input	**Wrapper**	**No Wrapper**
tennis.mpg	35118	40217
football.mpg	23143	27329
garden.mpg	38415	40273
salesm.mpg	57007	66578

CHAPTER 8. CONCLUSION AND FUTURE WORK

8.1 Conclusion

User behavior is poised to become an integral part of embedded system design methodologies. It is imperative to model user behavior for optimal utilization of system resources in mobile computing platforms. The rapid prominence of handheld devices and wearable computing only adds to the urgency of exploring new directions in embedded system design. The nature of mobile applications and multicore architectures further present complex design space for designers to navigate. Mobile application are highly interactive and there is increased demand on system resources during short bursts of usage. Heterogeneous multicore architectures present compute resources with multiple energy-delay profile to execute application threads. The choice of when to execute, what thread, on which compute resource to optimize a given performance metric is the problem of scheduling.

In this thesis, we have developed a simulation environment, PolyNoC, to aid in the exploration of heterogeneous multicore architectures for embedded system design. The applications in PolyNoC are designed to be polymorphic with the goals of expanding the energy-delay space from a few design points afforded by current multicore chips. A polymorphic scheduler in the operating system layer is tasked with morphism selection for application threads and uses a greedy scheme for maximization of user satisfaction at the application layer. We further propose an user satisfaction model using the sigmoid mathematical function to model the limitations of human audio visual system. This model drives resource allocation in PolyNoC with the goal of user satisfaction maximization. We extend user satisfaction model to the router layer for VC allocation heuristic. A uniform metric at the application layer and network layer removes conflicts in optimization goals.

8.2 Future Work

There are several directions to explore our proposed simulation environment and user satisfaction model. We have listed future projects associated with this thesis.

- Exploration of NoC architecture by changing the resources (number of CPU, FPGA) for a given set of morphisms of audio and video applications. The ideal set of compute resources to satisfy a given user satisfaction model will provide insight into allocation of silicon area to a specific class of compute resource.

- Metrics for mapping and scheduling application threads to compute cores are rife in network on chip research literature. Mapping and scheduling are treated separately since mapping is spatial and scheduling is temporal. However, from optimization perspective such as minimizing energy consumption, both issues have to be considered in conjunction. Drawing idea from relativistic physics, metrics which combine time and space can yield invariants which better optimize performance metrics. The interval in special relativity is an invariant, $dS^2 = dt^2 - dx^2 - dy^2$, for a two dimensional space transformation. dt is equivalent to scheduling and abstracts the temporal characteristics of a specific scheduling policy. dx and dy can represent the mapping of threads to the two dimesional space of compute cores in the NoC. It will be interesting to explore this metric idea.

- The sigmoid function is used as mathematical representation of user satisfaction. The shape of scaled Sigmoid ($\frac{1}{1+e^{-t}}$) resembles a cumulative distribution function. Differentiating the sigmoid gives the probability distribution function, $\frac{e^{-t}}{(1+e^{-t})^2}$. The shape of the pdf is very close to a bell shaped curve of normal distribution. User satisfaction model can be viewed as a cdf such that probability ($P(Y \leq y)$) that a given fraction of users (y) are satisfied for a specific resource allocation. Alternatively, the pdf may represent the distribution of the perception of HAVS for different resource allocations for various users. This is not surprising. Human physical characteristics such as height, weight follow bell shaped curve. A theoretical foundation for user satisfaction may emerge from these observations.

BIBLIOGRAPHY

[ANA04] D. Andrews, D. Niehaus, and P. Ashenden. Programming models for hybrid cpu/fpga chips. *IEEE Computer*, 37(1):118 – 120, Jan 2004.

[And] Android. Api guide. http://developer.android.com/guide/index.html, Accessed: 2015-06-17.

[App14] Inc Apple. App programming guide for ios. Sept 2014.

[BDB00] Vasanth Bala, Evelyn Duesterwald, and Sanjeev Banerjia. Dynamo: A transparent dynamic optimization system. *SIGPLAN Not.*, 35(5):1–12, May 2000.

[BM06] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38(1), Jun 2006.

[BWM+09] A. Banerjee, P.T. Wolkotte, R.D. Mullins, S.W. Moore, and G. J M Smit. An energy and performance exploration of network-on-chip architectures. *IEEE Transactions on Very Large Scale Integration Systems (VLSI)*, Vol. 17(3):319–329, March 2009.

[CFS] CFS. Inside the linux 2.6 completely fair scheduler. http://www.ibm.com/developerworks/library/l-completely-fair-scheduler/, Accessed: 2015-06-29.

[CM10a] Chen-Ling Chou and R. Marculescu. Designing heterogeneous embedded network-on-chip platforms with users in mind. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(9):1301 –1314, Sept 2010.

[CM10b] C.L. Chou and R. Marculescu. Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(1):78 –91, Jan 2010.

[Cud] Cuda. Cuda programming model. https://developer.nvidia.com/cuda-zone, Accessed: 2015-06-28.

[Dal] Dalvik. Dalvik virtual machine. https://source.android.com/devices/tech/dalvik/, Accessed: 2015-06-18.

[Dal90] W.J. Dally. Virtual-channel flow control. In *Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on*, pages 60–68, May 1990.

[Dal92] W.J. Dally. Virtual channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3(2):194–205, 1992.

[DT01] William J. Dally and Brian Towles. Route packets, not wires: On-chip inteconnection networks. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, pages 684–689, 2001.

[EDFL13] A. Eklund, P. Dufort, D. Forsberg, and S.M. LaConte. Medical image processing on the GPU - past, present and future. *Medical Image Analysis*, 17:1073–1094, 2013.

[HKKB13] Jan Heisswolf, Ralf König, Martin Kupper, and Jürgen Becker. Providing multiple hard latency and throughput guarantees for packet switching network-on-chip. *Elsevier Journal of Computer and Electrical Engineering*, Vol. 39(8):2603–2622, Nov 2013.

[HM04a] Jingcao Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 1, pages 234–239, Feb 2004.

[HM04b] Jingcao Hu and R. Marculescu. Energy aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, volume 1, pages 234–239, Feb 2004.

[HM05] Jingcao Hu and R. Marculescu. Energy and performance aware mapping for regular network-on-chip architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24(4):551–562, April 2005.

[HOM07] Ting Chun Huang, Umit Y. Ogras, and Radu Marculescu. Virtual channels planning for networks-on-chip. In *Proceedings of the 8th International Symposium on Quality Electronic Design*, pages 879–884, 2007.

[iVe] iVeia. Atlas i z7e. http://www.iveia.com/atlas-i-z7e, Accessed: 2015-06-12.

[KPT11] V. Krishnamurthy, S.D. Ponpandi, and A. Tyagi. A novel thread scheduler design for polymorphic embedded systems. In *Compilers, Architectures and Synthesis for Embedded Systems (CASES), 2011 Proceedings of the 14th International Conference on*, pages 75–84, Oct 2011.

[KPT14] Ka-Ming Keung, Swamy D. Ponpandi, and Akhilesh Tyagi. A placer for composable FPGA with 2d mesh network. *IJES*, 6(4):289–302, 2014.

[LCM+05] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, 40(6):190–200, Jun 2005.

[LSMW11] Weiguo Liu, B. Schmidt, and W. Muller-Wittig. Cuda-blastp: Accelerating blastp on cuda-enabled graphics hardware. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 8(6):1678–1684, Nov 2011.

[MCM+05] Csar A. M. Marcon, Ney Laert Vilar Calazans, Fernando Gehm Moraes, Altamiro Amadeu Susin, Igor M. Reis, and Fabiano Hessel. Exploring network-on-

chip mapping strategies: An energy and timing aware technique. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 502–507, April 2005.

[MNC+03] J.-Y. Mignolet, V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 986 – 991, 2003.

[Moo98] G.E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, Jan 1998.

[NAE+08] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal. Run-time management of a mpsoc containing fpga fabric tiles. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(1):24 –33, Jan 2008.

[NCV+03] V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Designing an operating system for a heterogeneous reconfigurable soc. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 22–26, April 2003.

[ND10] J. Nickolls and W.J. Dally. The gpu computing era. *Micro, IEEE*, 30(2):56–69, March 2010.

[Nvi] Nvidia. Tegra 4 series. http://www.nvidia.com/object/tegra-4-processor.html, Accessed: 2015-06-09.

[PAA+06] W. Peck, E. Anderson, J. Agron, J. Stevens, F. Baijot, and D. Andrews. Hthreads: A computational model for reconfigurable devices. In *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1–4, Aug 2006.

[PD01a] Li-Shiuan Peh and W.J. Dally. A delay model and speculative architecture for pipelined routers. In *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pages 255–266, 2001.

[PD01b] Li-Shiuan Peh and W.J. Dally. A delay model for router microarchitectures. *Micro, IEEE*, 21(1):26–34, Jan 2001.

[PT13] Swamy D. Ponpandi and Akhilesh Tyagi. User satisfaction aware routing decisions in NOC. In *Network on Chip Architectures, NoCArc '13, in conjunction with the 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46, Davis, CA, USA, December 7, 2013*, pages 11–16, 2013.

[PT14] Swamy D. Ponpandi and Akhilesh Tyagi. User satisfaction aware routing and energy modeling of polymorphic network on chip architecture. *Computers & Electrical Engineering*, 40(8):260–275, 2014.

[PZT13] S.D. Ponpandi, Zhang Zhang, and A. Tyagi. Polynoc - a polymorphic thread simulator for noc communication based embedded systems. In *Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on*, pages 1–8, Dec 2013.

[SAW+10] O. Shacham, O. Azizi, M. Wachs, W. Qadeer, Z. Asgar, K. Kelley, J.P. Stevenson, S. Richardson, M. Horowitz, B. Lee, A. Solomatnikov, and A. Firoozshahian. Rethinking digital design: Why design must change. *Micro, IEEE*, 30(6):9–24, Nov 2010.

[SKPK10] S. Sarkar, G.R. Kulkarni, P.P. Pande, and A. Kalyanaraman. Network-on-chip hardware accelerators for biological sequence alignment. *Computers, IEEE Transactions on*, 59(1):29–41, Jan 2010.

[SOM+08] A. Shye, B. Ozisikyilmaz, A. Mallik, G. Memik, P.A. Dinda, R.P. Dick, and A.N. Choudhary. Learning and leveraging the relationship between architecture-level measurements and individual user satisfaction. In *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pages 427 –438, Jun 2008.

[TI] TI. Texas instruments omap 4. http://www.ti.com/product/omap4430, Accessed: 2015-06-09.

[TLHC12] Wen-Chung Tsai, Ying-Cherng Lan, Yu-Hen Hu, and Sao-Jie Chen. Networks on chips: Structure and design methodologies. *Journal of Electrical and Computer Engineering*, 2012, Jan 2012.

[VZT03] M. Verderber, A. Zemva, and A. Trost. Hw/sw codesign of the mpeg-2 video decoder. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 7–14, April 2003.

[Xil] Xilinx. Zynq 7000. http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html, Accessed: 2015-06-12.

[ZPT14] Zhang Zhang, Swamy D. Ponpandi, and Akhilesh Tyagi. An evaluation of user satisfaction driven scheduling in a polymorphic embedded system. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, May 19-23, 2014*, pages 263–268, 2014.