Stony Brook University

OFFICIAL COPY

The official electronic file of this thesis or dissertation is maintained by the University Libraries on behalf of The Graduate School at Stony Brook University.

© All Rights Reserved by Author.

A Control Framework for Continuous Time Adaptation in Modern Day Embedded Systems

A Dissertation Presented

by

Sankalp Kallakuri

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

August 2007

Copyright © by
Sankalp Kallakuri
2007

Stony Brook University

The Graduate School

Sankalp Kallakuri

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of the dissertation.

Dr. Alex Doboli - Dissertation Advisor
Associate Professor,Department of Electrical and Computer Engineering

Dr. Wendy Tang - Chairperson of Defense
Associate Professor,Department of Electrical and Computer Engineering

Dr. Eugene Feinberg
Professor,Department of Applied Mathematics and Statistics

Dr. Sangjin Hong
Associate Professor,Department of Electrical and Computer Engineering

Dr. Radu Grosu
Assistant Professor,Department of Computer Science

This dissertation is accepted by the Graduate School.

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

A Control Framework for Continuous Time Adaptation in Modern Day Embedded Systems

by

Sankalp Kallakuri

Doctor of Philosophy

in

Electrical Engineering

Stony Brook Univeristy

2007

Modern day embedded systems require frequent changes in their archirecture due to the variations in the market requirements and the environment the embedded system is placed. For example, the data traffic that is seen by the embedded system may vary in amount and regularity. The quality requirements of the output may vary also vary. The variations impose constraints which cant be efficiently obtained by a static architecture and redesigning would severly affect time to market and cost due to over designed solutions. The over design comes in because the design may be done to handle all the varied scenarios and the full capabilities of the system would rarely be used. In a mobile computing/communication system power requirements are tantamount and there are number of schemes that used to control the voltage levels or alertness of these devices. There has been substantial work done in power conservation with the help of various control methods. We would like to extend some of these control methods

to the problem of controlling reconfigurable architectures in a dynamically varying environment.

The control methods started off with using discrete time models and then evolved to continuous time models for embedded systems. The continuous time methods have proven to give realistic representation of the dynamics of the system as compared to discrete time models. Another classification of the control methods can be done with respect to the deterministic and stochastic control frameworks. Deterministic control has been the classical approach used by engineers and computer scientists to device control methods due to the fact that most embedded systems are still synchronous and events occur at integral multiples of clock periods. The trend toward using continuous time methods has come about due to asynchronous operation and the fact that though the tasks may have discrete execution time values the values may exist over a large range and may follow a distribution in terms of their frequency of occurrence. Which led us to the usage of continuous time stochastic modeling for the embedded systems.

The control framework we use is based on CTMDP (Continuous Time Markov Decision Processes). These are a class of Dynamic Programming methods which implement stochastic control policies. These methods have been used for dynamic scheduling,load sharing and inventory control among other engineering and econometric problems. The beauty of these methods is that a queueing model of an architecture can be easily mapped to CTMDP framework. With appropriate rewards and constraints incorporated from the specifications of the application these methods throw the design problem as linear programming problem. We have studied cases where the problem may not be linear and methods have been developed so that the problem could be linearised.

The thesis aims at identifying the need of reconfigurability in different architectures and different applications. Such a study was necessary in order to systematically study the different design requirements and appropriate modifications in the control method. The embeddede architectures have beem broadly split into 3 subsystems by us , they are 1) the communication sub-system 2) the processing sub-system and 3) the memory sub-system. Untill now

we have concentrated on the communication and processing sub-systems. Designing the communication sub-system involved bus arbitration policies for synthesized bus architectures for real world applications. The stochastic arbitration policies were tested with other heuristics in a queueing model based simulation of the embedded architecture. They were compared on the basis on several performance metrics like buffer size, data loss, power conservation and delay. A design environment was developed in which the policies were sequentially used on a queueing model of the architecture and systematic data collection can be done and results can be plotted and analysed.

Experiments were performed for small test architectures as well as complex architectures for real world applications. The experiments were performed for a Hitachi SH DSP and an IBM Network processor. The synthesized bus architectures could be classified as architectures with and without redundant paths between processors as well as architectures with and without hierarchy. In terms of bus architectures hierarchy pertains to the existence of bridges that facilitate bus to bus communication instead of bus to processor environment. This introduction of bridges causes a master slave kind of environment which leads to a non linear problem definition making the problem intractable in the current frame work. One of the promising solutions to this problem seems to be to split the system into sections that could be modeled as a linear problem and then stitch the components together at higher hierarchical level.

In the processing sub-system experiments we initially tried a separate style of modeling in which the workloads were characterized by empirical curves obtained from the design specification required by the physical phenomenon being tracked. The curves were then split into windows where they were mapped bye parametrizable closed form functions. The second modeling method was to model the workloads as queueing requests. The communication sub-system optimizations led us to explore multiple suboptimal policies, which were optimal under certain conditions dictated by the workload. The policies were tested on cycle-count accurate as well queueing models of the architecture. The models were enhanced by making use of large and somewhat complex cycle-count accurate models as well as software profiling data in order to ob-

tain accurate service request rates and service provider utilizations.

In the future we shall try to dynamically manage the resources in the memory sub-system. The hardware resources available for computation can be classified into several configurations and the reconfiguration would involve selecting configurations in a continuously varying environment. We shall also extend the methodology to memory management. In hierarchical memories the management of caches and virtual memories provides opportunity for exploration of several trade offs between the cache management and eviction policies and the transactions caused by the instruction set over the platform. We would like to model the transactions as CTMCs and solve for optimal memory transaction patterns by using MDPs.

To my parents
Surg. Cdr. K. V. Suryanarayana and Dr. Mrs. Rekha Suryanarayana

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Modern day applications have high performance requirements in terms of power consumption, timing requirements, area constraints but there is a growing need to allow the designs to have flexibility in order to attain varied performance goals. Varied performance goals manifest themselves due to the differing environments the embedded applications created due to a dynamically changing market and available technology at a macro level and due to changing data traffic scenarios or power conservation schemes at a micro level. A large sector of EDA/CAD industry and academia is now concentrating on providing reconfigurability within the embedded systems they develope. The major advantage of having a reconfigurable system of course is that it can be provided as solution to a larger application domain as compared to static or non reconfigurable embedded systems, the second big advantage is that it can handle the need for dynamic architecture variation based on the variations of the environment the embedded system is in.

This chapter gives brief insight into the following topics

- outlines the need for reconfigurability in different embedded applications.
- gives a brief comparison the current methods that deal with reconfigurable architectures.
- explains why is system level evolution needed and why is it useful.
- use of CTMDPs in controlling reconfigurable architectures.
- our research incentives and contributions.
- organisation of this report.

1.1.1 Need for Reconfigurability

The embedded systems currently available deal with a vast range of applications and are placed in environments that vary hence demanding varied requirements from the embedded system [104][45].The levels at which reconfigurability can be provided is varied and different applictions demand reconfiguarbility at different levels of granularity, for example the smart memory systems developed in [59]. The work in reconfigurable architectures has been FPGA based and the earlier designs tended to be more RTL level reconfiguration rather than system level reconfiguration[70] [62]. The approaches though on programmable platforms like FPGAs were designed to tackle mostly static or slow reconfiguration. The order of the day though is to have run time reconfigurability [11] and it should be available at varied levels of granularity and different subsystems of the embedded solution [12]. The reason we want to stress in the reconfigurability in different subsystems is that the constraints and requirements of different systems within the reconfigurable embedded solution are different . We have considered three main subsystems

- Communication Sub System
- Processing Sub System
- Memory Sub System

Different applications like Nework Processors, Multimedia chips, Sensor Networks and Scientific Computing have different types and levels of reconfigurability requirements in order to satisfy contraints on buffer size,QoS,Delays,Data loss, Scalability and Power Conservation. A detailed discussion of reconfigurability requirements, types and levels of reconfigurability required in different applications has been presented in the second chapter.

1.1.2 Need for system level evolution and control

The system is made of several components each of which may or may not have a need to change with the environment the embedded system is in. Ideally the evolution of the system should be continuous allowing the system to be in optimal state of productivity, but there has to be a discretisation because the architecture of the system can change only in discrete steps. Though the changes are discrete the time taken between them should be continuous in order to allow the system to follow the real world environment variations closely. The representation of this evolution by a stochastic process naturally lends it continuous time variation. The reconfiguration is not only in the

architectures but to some extent also in the control policies themselves. As compared to RTL level reconfiguration and FPGA based reconfiguration this scheme is a more abstract and deals with the system as a whole. There have been attempts made at system level reconfiguration[80] [12] [11] but the control policies for these were either not discussed or were primitive when compared to the control policies discussed in this proposal.

1.1.3 Control methods for reconfigurable Architectures

Various control methods have been used for controlling various parameters in an embedded systems. The control methods can be classified in several ways some of which are

- discrete or continuous time methods[44][74]
- stochastic or deterministic methods[74][75]
- predictive or adaptive methods[52][110]

These methods have been compared and contrasted in this section. Most of these methods have been used in power manangement,memory management and data traffic management and hence fit well into the framework we are working in. The most basic policy which has been used is the Time out policy. This policy will keep the system or the component of the system in low power state, if the time till the next event will be such that the low power state will allow power saving, inspite of the power consumed in the transition to the low power state and the return to a more alert state [5] [6]. The timeout methods were discrete time methods with a threshold time beyond which it would be feasible was a fixed quantity [44]. The fact that these methods would put the system in the sleep state only after a certain threshold and wake up only after a certain threshold they would incur some penalty in the performance. In order to overcome these drawbacks predictive methods were developed. The predictive methods had either a static prediction or dynamic prediction. In both the cases essentially the correlation between the past power down times was used in order to predict the the time of next powerdown cycle. In stochastic control methods probabilities are defined over the states of the system as well as the possible actions that can be taken in these states[74] [75]. In general policies implemented have actions which could depend on a history of states and actions [24]. A more detailed discussion about the control methods has been given in the third chapter.

1.1.4 Use of CTMDPs in control of system evolution

The CTMDPs or Continuous Time Markov Decision Processes are a class of methods that form a subset of dynamic programming methods. They have been used in dynamic scheduling ,load sharing and power management problems among others. They allow us to define and prove mathematical optimality for the control policies developed which was not available with the heuristic methods. These processes can be reduced to Semi Markov or Discrete Time Markov processes for purpose of mathematical analysis. For a comprehensive discussion on Markov Decision processes and related issues we would refer you to [25]. We will refer to the use of these methods in power management a lot in this prelim as substantial work has been done in adapting these methods to SoC power management [10][76]. We have explored some improvement in these methods which have not been tried in the power management scenario as yet, we are also trying to elegantly extend this theory to problems which do not fit well into the CTMDP framework.

1.1.5 Contributions and Incentives

One of the incentives for pursuing this problem was the ongoing research in Bus Synthesis in our research group [99] [100]. The Bus Synthesis effort resulted in layout aware bus architectures which were sensitive to the communication load as well as the floor planning of the embedded system. The next natural step in designing the communication subsystem after the obtaining the bus architecture was designing arbitration policies for the buses. The continuous time nature of the tasks and traffic in SoCs led us toward using continuous time stochastic models. We realized that use of such models also lends itself well to the application of stochastic optimisation methods like CTMDPs,which we then used and would like to further explore. An incentive for pursuing the possible continuous time adaptation in the processing subsystem was again due to ongoing research in our group for real time systems and sensor networks [107] [108]. This effort deals with making differing sets of HW resources vailable to match timing constraints. We would like to explore the optimisation of this process.The third incentive was that the CTMDP methods allow us to analyse in a mathematical framework the optimality of the generated policies for control.

The novel contributions in this report are:

- a systematic study of scenarios in applications and architectures that would require continuous architectural adaptation as discussed in Chapter 2.

- the use of continuous time stochastic models in the design of arbitration policies and buffer allocation for Communication Subsystem of an SoC as discussed in Chapter 4. Especially the formulation of the arbitration problem for finite queues in the presence of redundant servers and hierarchy in the queueing network.
- extension of the continuous time adaptation modeling to the Processing Sub System as discussed in Chapter 5.
- studying the use of multiple sub-optimal policies in an environment where the requirements vary drastically. Sensor nodes provide such a case study as put forward in Chapter 6.
- improvement to the request rate generation models for transaction level simulators as shown in Chapter 7.

1.1.6 Organisation of this Preliminary Proposal

The following chapters of this report are as follows :
Chapter(2) deals with a detailed study of the need for continuous time adaptation in different architectures while they are utilised in different applications, chapter(3) is a detailed comparison of control methods , chapter(4) is an explanation of CTMDps as well as a discussion of the experiments and results, chapter (5)is an extension to the continuous time modeling effort (6) is a case study of sensor network nodes which need reconfiguration and exist in environments which may require very different levels of computation, chapter (7) is the future directions of the research effort and chapter (8) is the conclusion.

Chapter 2

Continuous Time Adaptation in Embedded Systems

2.1 Introduction

In this chapter, a study of different embedded architectures and applications in terms of requirements for continuous evolution.

The following table has been discussed in detail with respect to the requirements in the industry for some applications and how the static architectures fall short and what dynamic improvements are required.The discussion is split over the two domains of architectures and applications and current reconfiguration requirements and reconfiguration abilities have been discussed. The difference between current reconfiguration concepts and schemes and the ones used by us is that the current methods are usually deterministic and based on discrete time steps [6][13]. Another difference between our approach and the current methods is that at whatever granularity the changes take place in the actual architecture, they could be efficiently abstracted into the state space of a CTMDP[10]. The feasibility and need of the control methods we propose may be limited in some scenarios due to pre existing infra structure in a particular sector, hence it is imperative to classify the sectors where these methods would find use.

2.2 Detailed Discussion

The performance requirements placed by applications on architectures vary with the applications. The suitableness of an architecture is judged by how well it satisfies these performance requirements. The differences in the performance requirements and the resources that can be provided by an architecture is what causes the performance of a certain product to go down and creates

requirements for new architectures for example a change of protocol for a mobile communication chip. In case a requirement for a different architectural resources presents itself sometimes it is faster and cheaper to have the resources available and they can be adapted for different usage at run time in a continuous fashion. We have studied the following applications and enumerated the pertinent performance requirements. The applications are mobile computing, sensor networks, scientific computing and biomedical applications. We have also studied how well these applications get mapped to certain architectures as depicted in Figure 2.2. In some cases the need for continuous time adaptation is felt and in some there it hasn't been pertinent both these situations have been discussed in deatil. There are certain cases where there has been no discussion because of a large mismatch between the requirements of the application and the resources the architecture can provide.

2.2.1 Requirements,Constraints and Trade offs in Mobile Computing

Mobile computing no longer deals with just voice or text. With the advent of PDAs,laptops and cellular phones equipped with cameras the traffic has changed to more advanced applications like web browsing,email and audio/video decoding [103][95][94]. The communication between the cores or modules in a mobile computing SoC solution would involve heavy amount of traffic flowing through the SoC due to such high end applications [102].

- *Buffer Space*: The availability of buffer space is a problem because the space available on chip is very limited compared to regular networks. Thus efforts have been made to have optimal buffer size in presence of video traffic [102].

- *Bus Structure*: Apart from buffer space the availability of buses is also an issue and developing efficient communication subsystems that can handle the on-chip traffic is an important issue as discussed in [53] [29] and [117].

Apart from having a very efficient communication sub system the Mobile Computing applications need to have a processing sub system which can provide error free data after it has been decoded from the modulated signal.

- *QoS-Performance Trade offs*: The requirements on the processing sub system are that it should be able to keep the bit error rate low and keep the performance within some specified QoS level. Trade offs between the

QoS and performance are used in order to have maximum utilisation of processing power. For example in a wireless environment where there is fading and the inclusion of an extra finger in the rake receiver can help keep the bit error rate low but in turn increase the power consumption [83][2].

2.2.2 Requirements, Constraints and Trade offs in Sensor Networks

Sensor networks have very different constraints and requirements as compared to real networks or other computing applications as shown in [37] [88] . The idea of sensor networks is to have a small amount of processing done to the data and then to stream the data on the network.

- *Space and Power Constraints*: Major constraints for the sensor nodes are space and power consumption. Thus the processing subsystem does not need to be very advanced and usually the sensor node architectures are microcontroller or DSP based. The nodes have to be small as large numbers have to be cheaply made available in order to monitor vast spaces. The OS and the controllers have to be light weight and highly customised to the application[96].

- *Fast and Light*: In sensor networks one of the trade offs is between the on board processing and real time data flow. The aim of the sensor networks is to have minimal amount of processing and provide samples of the environment being monitored in real time. This calls for optimal amounts of storage and processing power being made available on a sensor node[43] .

- *Trade offs in Hardware for Control and Processing*: Another trade off in sensor networks is between using a single processor which is shared among the control and data processing or to use separate processors which would take care of the data processing and the control. The idea is that the shared processor will take less space and on chip communications would be lesser than the two processor solution which is the default in current systems, for example the MICA motes [115], but the independent processors will be easy to build and are already available as sensor node architectures.

- *Varying Connectivity requirements*: The sensor network will sample a set of parameters in the environment it is placed and contain some spatial

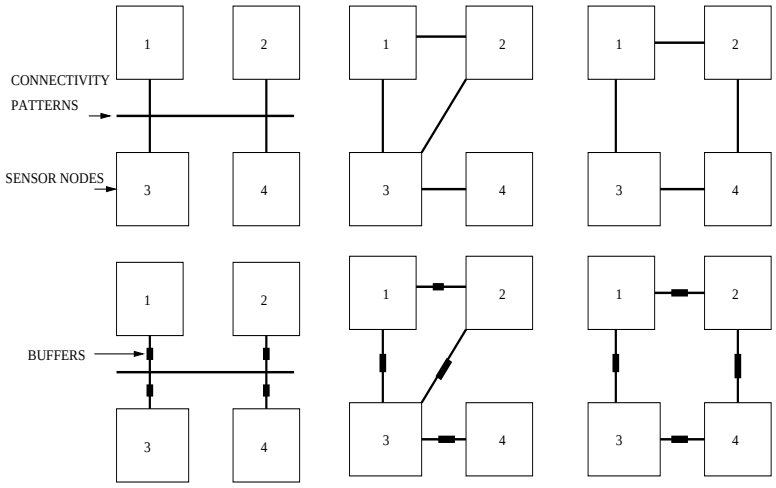

Figure 2.1: **Adaptation in node storage due to variation in network topology**

representation of the environment. In case the data collected from the network is analysed offline and temporal variations of the parameters in the space monitored by the network are of no consequence then synchronisation between the nodes need not be done but this usually doesn't happen [119] [20]. In case the variation of the monitored space over time is necessary then the need for synchronisation would be neccessary or at least the data collected would have to be tagged with time of collection so that corresponding samples of the space can be assimilated[119]. The communication may not involve a central base station so the way in which data packets move about in the network would require a flexible communication and storage subsystem and the spatial knowledge about the temporal status of the network would lie in the community of nodes and not in any particular node. This may cause variable connectivity patterns between the sensor nodes and thus will affect the storage and communication subsystems on the nodes due to the redistribution of traffic as shown in Figure 2.1. Since the sensor nodes may communicate by wireless the connectivity patterns could change dynamically. The need to store and process data would be dynamically varying and the buffers placement in the network topology and the size of the buffers would have to be varied dynamically.

2.2.3 Requirements,Constraints and Trade offs in Scientific Computing

Scientific computing has two basic approaches one is to use supercomputers which have huge amount of computing resources or to use clusters of smaller computers. An example of a large monolithic supercomputer is a Cray X-MP from Cray inc. [14] and Earth Simulator from NEC [19] which is capable of 35.61 TFLOPs.

- *Load Sharing and Management*: The requirements in these sytems are that the load of the task should be equally shared among the processors in order to have the fastest execution of the task. Another requirement is to have an efficient management of queues and stacks.

- *Memory Requirements*: In many cases the memory available turns out to be a constraint for applications to be run on supercomputers and while designing them memory requirments of possibly a decade in the future are provided [51].

- *Monolithic versus Distributed*:There exists a trade off though between the two approaches. In the distributed computing case where there are several general purpose processors interacting ther will be need for network processing systems like routers and switches to handle the communication whereas the communication in the monolithic machines would be more like a GPP but at a large scale. In case the computation is distributed over several smaller computers then the communication time betweeen the inidvidual processors could be significant and would have to be taken into consideration [72].

2.2.4 Requirements,Constraints and Trade offs in Biomedical Applications

Biomedical applications involve robust and fault tolerant design [58]. Most biomedical systems have very robust design.

- *Fault Tolerance and Slow Degradation* : They have slow degradation of resources in certain systems which must not fault suddenly and yet should be able to sustain very long battery lifes. These are devices like hearing aids,pacemakers and defibrillators. They usually have very low power consumption and as some of them may be internal to the human body the prospect of replacement is avoided as it would involve surgery.

- *Real Time Monitoring*: The Monitoring of biomedical signals involves real time processing of the incoming signals. Accurate display and analysis of the sigals is required in case of life support systems which have to monitor a patient and accurately display the vital signals as well as comprehend situations which require alarms to go off [41]. The system apart from being fault tolerant should be able to commmunicate in real time with sensors and displays.

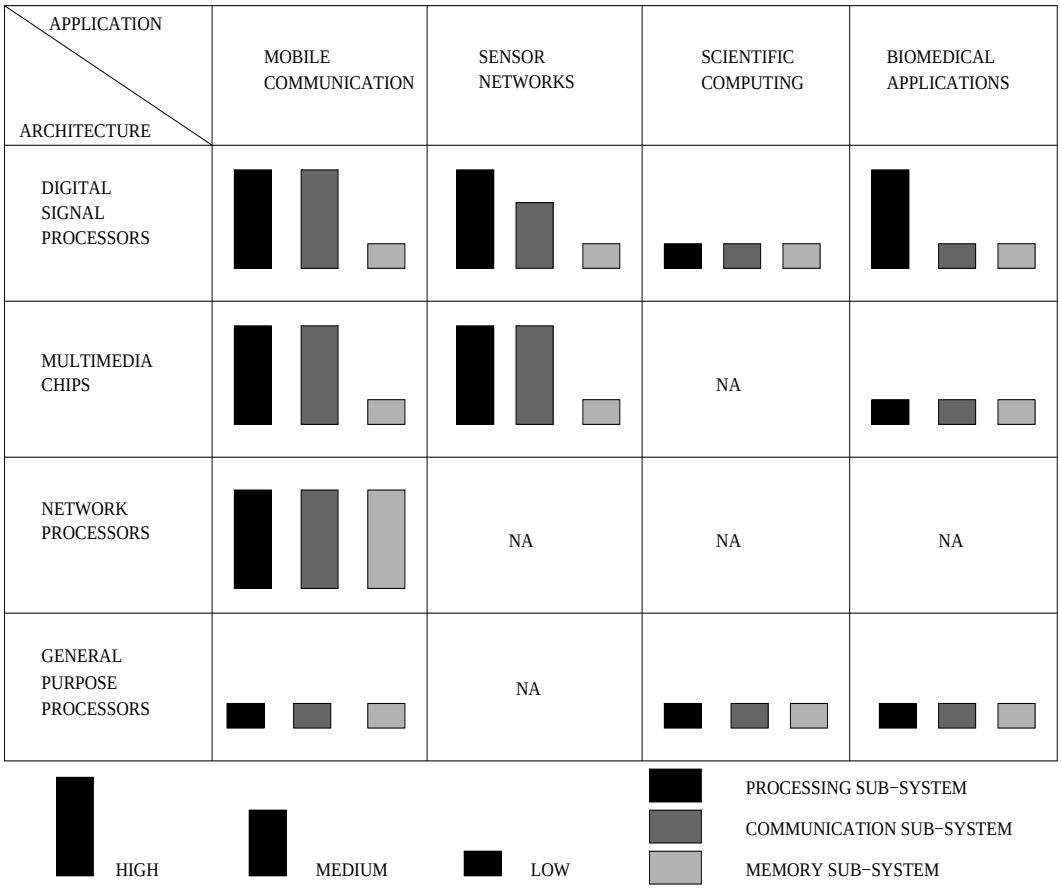

Figure 2.2: **Evolution requirements in embedded applications and architectures**

2.2.5 Continuous Time Reconfiguration in DSPs for Mobile Computing

The use of DSPs in mobile computing is due to their ability to execute signal processing operations like filtering ,equalisation ,FFTs and algorithms like Viterbi decoding[97]. The DSPs have an ability to perform high precision floating point arithmetic at high speeds due to availability of multiple processing elements or in some cases multiple datapaths. They have circular buffers to perform convolution which is a basic operation in many signal processing applications. To show the high amount of parallelism available in a DSP given below is an example of the datapath of the C64x DSP[97]. The C64x has two datapaths each with 4 processing units and a 32x32 bit register file as shown in Figure 2.3. Each register file has 6 write ports and 10 read ports which enable parallel read/write operations of operands/results from all processing units. Several operations can be executed in parallel like six 32-bit additions, four 16-bit multiplies,eight 10-bit multiplies and two 8-bit Galois field multiplies [1].

The DSPs used in mobile computing are available in three basic flavours 1) Application Specific, 2) Domain Specific and 3) General Purpose [27]. The application specific have highly customised datapaths and are built for highly specific tasks like speech decoding [66].They have a heavy initial cost due to the high application specific nature. The domain specific have a smaller time to market and cost as compared to the application specific as they are specialised to perform certain sub tasks like equalisation or viterbi decoding [67] [26], e.g Tic540 and TCSi Lode. They can be used for other purposes too, but are designed with a certain application or set of applications in mind.The general purpose are the ones prevalent in currently used devices. They have very small time to market but lack in performance compared to the domain specific DSP's.

- *Trade Offs in DSPs*: With reconfigurable DSPs the trade off between time to market and performance could be reduced by using the reconfigurability to adapt to the the new requirements instead of redesigning the chip, in other words one could say that with reconfigurability the domain specific DSPs could also handle the applications of the application specific DSPs. Apart from the changing standards set by the market the chips are also sensitive to QoS like parameters where they could use more computational resources in case the output accuracy requirements get more stringent [2] [80].

- *Reconfiguration Requirements* : The reconfigurabilty requirements and

possibilities in these DSPs can be seen at all the three subsystems though more in the processing and communication subsystem than in the memory subsystem. The processing subsystem is the one that goes through a change depending on the QoS requirement and its changes cause changes in the communication subsystem too, for example in [80] we see that the number of fingers in a Rake receiver could be increased if the bit error rate has to be reduced. In the Chameleon RCP by [80] the processing subsystem is highly reconfigurable with several processing units each reconfigurable within itself to operate on data types of variable length. This demands reconfigurability in the memory subsystem too. The various constraints that are expected to manifest themselves in such a adaptive scenario would be time bound distribution of data in the communication subsystem and minimal resource utilisation in the processing subsystem. The memory subsystem is expected to have the least change requirements and hence is being left alone for this sector of the application/architecture space defined in Figure 2.2.

The demand of the current scenario to change the from one configuration to another in a single clock cycle for the Chameleon RCP requires the alternate configuration to be stored and then used when the need be. This kind of reconfiguration may be needed in cases where there is a fundamental change in the protocol and the processing apparatus at a base station needs to go through an automatic change to handle the update in the protocol. This kind of reconfiguration would need very hard timing constraints on the system. Though the reconfiguration is very quick the flaw with this methodology is that there can be only a limited number of alternate configuration of the system which is stored and the decision as to what this alternate configuration would have to be decided by the designer. In the control methodology we propose the change to an alternate configuration could still be made in a clock cycle but to with a changing environment the system shall be able to choose a different configuration without designer intervention so it may take a couple of more clock cycles to reach the optimal configuration but there need not be any designer intervention.

The reconfiguration requirements and possibilities in the subsystems have been depicted in 2.2 and their relative degrees of adaptation have been depicted by the height of the graphs.

2.2.6 Continuous Time Reconfiguration for DSPs in Sensor Networks

As sensor networks are relatively new the possible use of DSPs in them has not been explored extensively [105]. The use of DSPs in doing energy efficient data transfer has been explored. The data transfer is energy efficient because the node performs a fourier transform on the data or does an on site analysis of the data and then transmits it. This is done because it is easy to scale or threshold the fourier components and transmit minimal amount of components. The data is obtained by using inverse tranforms at the receiving end. The number of sensing elements in the network may be very large and the networks life would have to be extended to its limit. Thus making power conservation an issue. Another issue with a large number of nodes is the data storage and communication as the nodes may need to send data directly to other nodes and there may not be a central base station raising many communication and storage issues [28].

The use of DSPs is primarily going to be in the data processing section of the sensor nodes as the sensors will be dealing with an analog environment and the Analog to Digital Converters shall be interfaced with the DSPs which shall take care of the sampled digital output. Though the need or type of evolution that would be required in a DSP based architecture for a sensor network would depend a lot on what the sensor network is being used for, we can at this moment make some comment on the possible requirements.

Continuous time adaptation will be needed in the buffers used to store the sent and received data as their environment changes. DSPs usually have very advanced proccessing subsystems as shown in 2.3 but their communications subsystems aren't very advanced in the sense they usually do not have ability to handle different protocols and packet lengths and are usually dependent on a coproccessor like a GPP or a microcontroller to handle the external communication [69]. The DSPs communicate with peripherals like A/D converters and memories under the control of the coproccessor. Thus adaptation of the communication sub system of the DSP would be needed but a better solution than changing the DSPs already specialised architecture would be to allow the coprocessor to handle the adaptation needed for a sensor node.

The reconfiguration requirements have been shown in 2.2. The major reconfiguration requirements are in the processing subsystem.

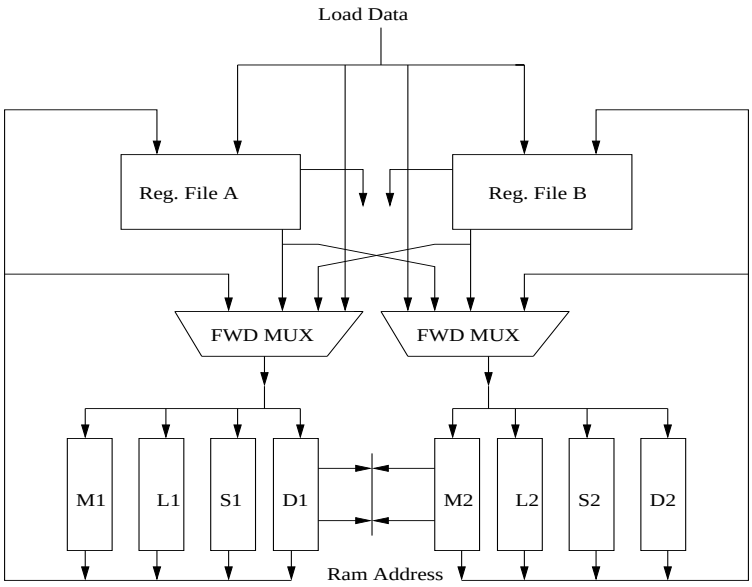

Figure 2.3: **Datapath of C64x DSP [97]**

2.2.7 Continuous Time Reconfiguration for DSPs in Scientific Computing

DSPs are used in many scientific applications like geology,hydrology and astronomy. The reason DSPs are used is that they have some features not available in other architectures like multiple bus architectures,multiple processing units which can function in parallel. They can usually perform multiple read/write operations simultaneuously from I/O or memory. They may have multiple DMA channels for high speed data movement without processor interference, special addressing modes with circular or bit reversed addresses and instructions that could handle normalisation and saturation [98] [30].

The applications are essentially imaging based and involve very data intensive work which cannot be handled by single proccesors available in the market now. The work being data intensive there is a need for splitting the tasks among several processors. Instead of alloting different tasks to different processors the approach here is to split the acquired data over several processors and the the output is then combined together. These applications do not require very high reconfigurability but do require a high amount of parallelism. The incoming data may be split across several DSPs to speed up the processing especially in SAR(Synthetic Aperture Radar) in which there are several small aperture radars which have there images stitched together[49] [55].

Continuous reconfigurability is a requirement of systems where the envi-

ronment or application puts varying demands which isn't seen in these applications. The output usually has to have very high accuracy and resolution and trade offs are not tolerated, so a brute force method is used, in which the highest requirement of computing power has to be provided. The low adaptation requirements have been shown in 2.2.

2.2.8 Continuous Time Reconfiguration for DSPs in Biomedical Applications

Many biomedical applications involve DSP based computation and we have examined the need for continuous time online adaptation in these applications. The DSPs have been used in varied biomedical applications like hearing aids and monitoring biomedical signals [58]. The use of continuous time architectural adaptation seems minimal in these applications but there are a few which need it.

There is a need for architectural reconfiguration in hearing aids [61]. The ability to reconfigure the device will allow better speech encoding decoding algorithms to be run on the same device and there wont be need to design a new hearing aid. There is a need for power management for devices like hearing aids which run on batteries but this has been achieved through power management policies like using varied voltage levels and very low current designs [63]. The possibility of exploring trade offs in fault tolerance and power consumption in the hearing aids has been explored too. Though fault tolerance is very important for biomedical systems, especially for critical life support systems but for hearing aids a slight degradation in performence could be tolerated for the sake of giving the device a longer battery life by reducing the power consumption [58]. The adaptation requirements are only in the processing subsytem and the memory subsystem and communication subsytem are expected to be static 2.2.

2.2.9 Continuous Time Reconfiguration for Multimedia Chips in Mobile Communication

Multimedia chips are becoming a necessity in mobile computing devices with still picture and video clip transmission becoming the need of the day. The need for reconfigurability is heavy in these applications as the available bandwidth decides the quality of the picture that is being sent.

- *Buffer Space dependance on Channel Speed*: The slower channels would have to accumulate the video transmission in a buffer before displaying

the video as the display would require a continuous stream of frames which may not be available in a slow channel. This variable level of resolution in the video quality and the buffering would impose varying constraints on the processing subsystem and the communication subsystem hence there is a need for continuous adaptation.

- *Qos Based Adaptation*: The need for continuous time reconfigurability also manifests itself due to the possible data shaping that shall occur and the different data shaping schemes will need a dynamic buffer allocation technique [94] . The ORbit scheme developed in [95] is a possible solution to this problem.In this method the video traffic is not recieved in a temporally ordered sliding window fashion but the frames arrive out of order and are stored in their compressed form. Then these frames are compared with some of the frames that special frames and are sent across due to their content which would help reconstruct the video clip inspite of jitter in the video stream. This method of sending some key frames with their temporal time stamp in advance and then allowing out of order buffering causes variation in the buffer lengths because the buffer lengths can be used as constraints for the number of ORBit type frames to be stored for a certain tolerance in the video quality as shown in Figure 2.4. Thus allowing a trade off between the video PSNR and the number of frames to be stored.

Though the ORbit method seems efficient and is claimed to be faster than some GA based methods, we would like to test its optimality when compared to CTMDP based methods discussed in later sections of this report. The reason we feel the CTMDP methods could outperform the ORbit method is that we can obtain theoretical or mathematical optimality in a control policy obtained throught the CTMDP methods, whereas in the ORbit method appears to have originated due to the modelling of the distortion due to frame based jitter in video traffic. Apart from this, the buffer allocation should be done not just based on PSNR of the video but also taking into account other parameters like the on chip bus structure and the traffic patterns that would occur on it, which again leads us toward multiobjective CTMDP based optimisation for continuous time reconfigurability. The adaptation requirements of the subsystems have been shown in 2.2, the communication and processing subsystems require high levels of continuous adaptation.

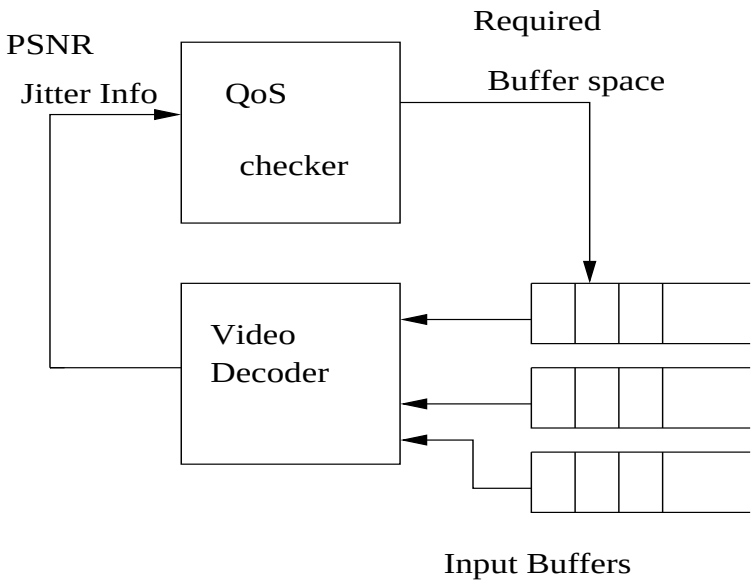

Figure 2.4: **QoS based adaptation**

2.2.10 Continuous Time Reconfiguration for Multimedia Chips in Sensor Networks

In sensor networks the data accumulated from the environment has to be processed and presented in a easily understandable mannner. The data should be easy to visualise and analyse so that a technician or other reactive embedded system could act accordingly when it receives the data. In sensor networks especially multi hop networks where data packets could arrive through different paths and there is no central base station there is an issue of which packets to keep and which to discard [92] [93] in terms of preserving video quality.

- *Bandwidth versus Resolution*: Another issue is when and by how much the resolution of a video stream should be reduced. The need for this reduction in resolution comes about due to a large number of streams being needed to have a complete spatial representation which means very large bandwidth requirements. So in order to have a complete representation yet use a lower bandwidth the streams could be selectively degraded in their resolution or scalable compression could be used [89].In order to have this variable resolution and selective data acceptance as in Figure 2.4 there will be need for reconfigurability in the processing sub-system and the communication sub-system.

- *Mixed Traffic Environments*: The continuous time adaptation would be needed due to the varied video quality as well as the possible coexistence of other media like sound and text and processing resources and buffer requiremnets will vary. Characterisation of these variations has been studied in [102],a stochastic modeling technique that links and compares the data network traffic with the on chip traffic has been presented in order to study the buffer space requirements. An example of a mixture of wireless traffic has been discussed in [101]. It shows a wireless military environment useful for tracking and shooting down hostile incoming missiles or planes. It involves airborne,seabourne and land based surveilance of an area. Due to the different geography channels chracteristics for transmission are different thus different communication protocols are used but they have to be coherent to the sensors in the different regions of the war theatre.

Adaptation in multimedia chips for sensor networks will be felt mainly in the communication subsystem and the processing subsystem as shown in Figure 2.2 due to the varying QoS requirements as well as the mixed traffic scenarios.

2.2.11 Continuous Time Reconfiguration for Multimedia Chips in Biomedical Applications

Many biomedical applications involve video and audio data processing[71]. The need for multimedia chips to handle images with high resolution and sounds of very low amplitudes and high frequencies has been felt. One such application is virtual colonoscopy which involves 3D Volume rendering of the colon and is a non invavsive method of diagnosing colon related ailments cancer bein the major one. The regular chips cannot handle the resolution requirements and some architectures [34] [71] have been developed which allow parallel data flow and have fixed bandwidth interconnects between the processors to aid 3D visualisation of data from MRI ,CT Scan, PET,SPECT like biomedical imaging techniques. The ability to paralelise the 3d visualisation comes from ray tracing methods.

Though like other architectures used in biomedical engineering these have limited capabilities and need in terms of reconfiguration because is almost always done keeping in mind worst case scenarios and reconfiguration is not needed or implemented as shown in Figure 2.2.

2.2.12 Continuous Time Reconfiguration for Network Processors in Mobile Computing

Base stations for wireless routers need network processors. The tasks implemented by the Network procesor are to identify packets in terms of their size,content,protocol and destination. To manage the flow i.e to do some queueing management on the packets. To route the packets, which would mean a switch which could deal with variable length packets is required. Most of the tasks from the Network Layer of the OSI Data Networks stack including fragmentation, regrouping and security are implemented in network processors [81].

Dynamically configurable network processors are now available in the market [80] and their abilities are being tested as compared to static architectures [101]. The reconfigurable can adapt to the varying traffic environment including varying traffic intensity and varying packet types[101].These architectures have a very high reconfigurability in their processing subsystem as seen in [80].The need for reconfigurability comes about due to the various different protocols and packet lengths that a base station could be encountering in a wireless environment. The varied packet types and protocols that could coexist in an environment will cause the architectural requirements to change drastically depending on the protocol being encountered at a given instant. The CS2000 from Chameleon systems is such a reconfigurable network processor but we feel the control methodology for the reconfiguration is primitive and basically uses an on /off type control whereas dynamic reconfigurability should be able to capture more detailed and smooth architectural reconfiguration.

Efforts have been made to characterize the data traffice characterisics with statistical methods traces of internet traffic are available and have been analysed for the type of traffic in terms of the types of protocols being used ,packet sizes and media [15]. This type of analysis would fit in very well with making decisions regarding the dynamic adaptation of a network processor in terms of the switching fabric and queue management system as shown in 2.5. The adaptation requirements in all three sub-systems have been shown in Figure 2.2.

2.2.13 Continuous Time Reconfiguration for General Purpose Processors in Mobile Computing

This section deals with the use of general purpose processors in mobile computation. We also try to examine if there is a need for reconfigurability and would it be feasible to provide it. As applications get more and more complex

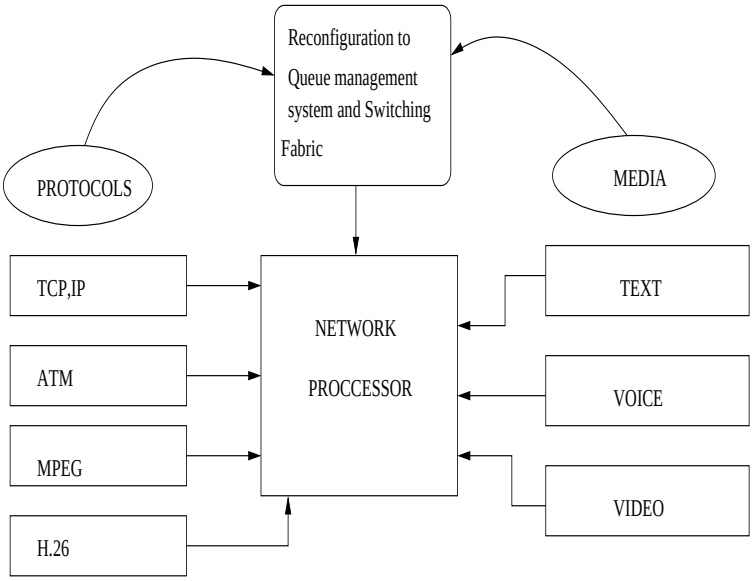

Figure 2.5: **Factors affecting adaptation in a Network Processor**

the cost to build ASIC solutions for them increases. This has brought about a new school of thought which aims implementing extremely complex tasks on general purpose processors in order to save on the initial capital investment which is high in ASICs. The general purpose proccessors are used in mobile devices like PDAs and laptops since power saving is the first priority these devices have some reconfigurability present in them to do power management but architectural reconfiguration has not been explored for GPPs. The scope of architectural reconfiguration in GPPs is minimal as most of the processing is left to the software and there is no application specific hardware. One of the reasons there is no architectural reconfiguration in the GPPs is that they are usually interfaced with a DSP as a coprocessor when used in Mobile Computing applications [69] [84] and the reconfiguration if any would be in the coprocessor rather than the GPP as reflected in 2.2.

2.2.14 Continuous Time Reconfiguration for General Purpose Processors in Biomedical Applications

GPPs are used in biomedical engineering for image proccessisng applications like MRI,Spect and PET but they are used to run complex statistical or signal processing like algorithms which would be too complex and expensive for ASIC implementation. Some small offline enhancements to GPPs have been

studied for imaging applications and show that the flexibility offered by the GPP may be useful as ASIC architectures may have components that lie idle for many instructions and cost in area. Whereas the GPP will be using a small pool of processing power which will have a high utilisation [90]. The need for continuous time adaptation of the architectures for GPPs in Biomedical systems has not been felt,the reason being that the applications do not encounter traffic based or QoS based variations which can be explored for adaptation in the architectures. GPPS by themselves aren't customised towards imaging applications but the flexibility they offer is useful and efforts have been made to get hybrid systems with multiple processors which are able to incorporate some of the GPP features [87].

2.2.15 Continuous Time Reconfiguration for General Purpose Processors in Sensor Networks

GPPs may also see use in sensor networks as they offer flexibility and less effort to design and produce. At this moment though there is only speculation and most sensor node architectures are DSP or microcontroller based [84]. The architectures are DSP or microcontroller based due to the space and power consumption constraints met by them. DSPs and microcontrollers have more application specific architectures compared to GPPS,hence provide better performance for the same or lesser area compared to a GPP. Another disadvantage of using GPPs is that they have very complicated instruction sets and operating systems which aren't necesssary for the low processing requirements of a sensor node [96]. Since the use of GPPs hasnt been explored yet the possibilty of their being need for continuous time adaptation in GPPs for Sensor Networks doesnt make sense at this point in time. Though in order to visualise the data provided by the network or for analysis of the data by a technician there could be need for a GPP.

2.3 Summary

In this section we studied various applications and architectures which may or may not need continuous time reconfigurability. Some observations are that applications with high communication are the ones which need the maximum amount of architectural reconfigurability for example mobile computing and sensor networks. The reconfigurability we have discussed is not only needed at run time but is needed over a more detailed and varied granularity level as compared to the current FPGA based reconfigurable computing community. For

such a flavour of reconfigurability the control methods have to allow for more fexibility and should be able to match the architecture to the requirements in a more optimal fashion. In the figure 2.2 we have presented a comprehensive view of the possible continuous time reconfiguration that would be needed in the various architectures depending on which application they are used in.The DSPs used in Mobile Computing and Sensor Networks have a need for continuous time adaptation in there communication as well as processing subsystems while the ones used in Scientific Computing do not see much adaptation. The DSPs used in Biomedical applications need some adaptation especially if slow architectural degradation is required. The multimedia chips used in Mobile Computation and in Sensor Networks need adaptation in the processing and communication subsystems with the ones used for Sensor Network applications need reconfigurability in the memory susbystem also. The network processors need reconfigurability in all three subsystems but can be used in only the Mobile Computing applications. The General Purpose Processors have minimal need for adaptation but find use in almost all applications as shown in figure 2.2. The next section will deal with existing control methods and possible new methods that could be used.

Chapter 3

Control Methods for Reconfigurable Embedded Systems

3.1 Introduction

In this chapter we present a discussion of the control methods used in embedded systems.

In embedded system design qualitative as well as quantitative performance evaluation methods have been extensively used [76] [68]. The disadvantage of qualitative performance evaluation techniques is its difficulty to capture any dynamic behavior like task scheduling and arbitration. For example,the quality of hardware/software partitioning is traditionally evaluated by a cost function which models number of synchronization points, number of interactions in handshakes between processes, duration of idle periods etc [112]. The parameters describing the above mentioned attributes of a system in the cost function do not vary dynamically, thus making it difficult to calculate system latencies and power consumption. In order to capture the variations in these attributes there is a need for flexible models, which incorporate arbitration and scheduling details, and can accurately track the state of the system. Quantitative performance evaluation relies on simulating the system for a sufficiently long period of time with input data, which possesses the required characteristics for accurately modeling the system [54]. Quantitative values can be accurately obtained for the performance attributes of a system. Simulation based performance evaluation has the drawback of long simulation times in case of large systems with less information known prior to the simulation.

During system design the constraints that are needed to be maintained while defining a policy are obtained by the simulator in its exploration loop. We quantitatively find certain parameters of the architectures using stochastic models, and then simulate the stochastic process that underlies the function-

ing of those architectures. Many problems in architecture selection, power estimation, and power management can be studied by stochastic simulation methods. The various architectures have been modeled as networks of queues with the elements of the architecture modeled as service providers, service requesters and service queues [76]. The advantages of using stochastic simulation in context of embedded systems are abstraction of gate level and RTL level characteristics into the system level, as well as capturing interactions between between parameters like delay, queue lengths and power consumption which may not be caught while doing qualitative analysis or simulating deterministic models [68].

The control methods have been studied under the following classifications.

- discrete or continuous time methods [44][74]
- stochastic or deterministic methods [74][75]
- adaptive predictive or static predictive methods [52][110]

3.1.1 Discrete and Continuous time methods

The initial efforts in control methods were based on discrete time policies where typically the control would be at RTL level and thus closely linked with the clock period [31] [9]. With system level design the time taken by tasks on processors are no longer discrete and have values over continuous ranges and these values have certain distributions. As an example we plotted the time taken by the gzip program for zipping files of the same size but with different amounts of variance in the data. We found some interesting characteristics from the plot 3.1. We found that there exists a range of values over which the execution time exists and there exists a trend in the distribution of execution times. Hence the reconfiguration should take place in continuous time in order to match the variation in the execution time on tasks. The discrete time method in [68] uses a Discrete Time Markov Process to model the events in an embedded system. These aren't as accurate as the Continuous Time Makov Chain(CTMC) models used in [74] [75] [76]. In the CTMC models the transitions between states have a finite transition time associated with them and this is closer to the real world situation as compared to the discrete time methods in which the transition between states is instantaneous.

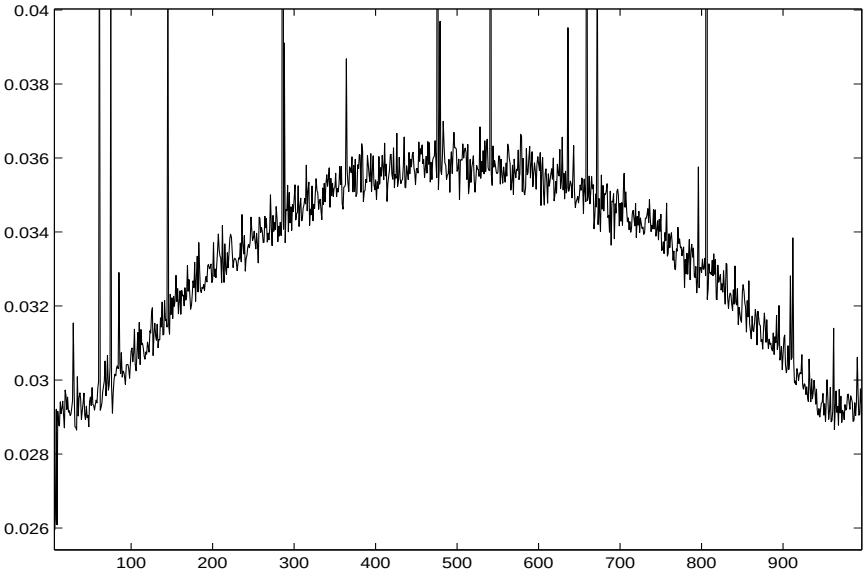

Figure 3.1: **Distribution of execution times for gzip program on 1000 byte files with varying no of "ones" in the 1000 bytes**

3.1.2 Stochastic and Deterministic Methods

Another set of competing approaches toward control of embedded systems are deterministic and stochastic methods [74][6]. The idea behind stochastic methods is to have decisions that are taken with some probability rather than with complete certainity under the influence of some stimuli. In modern embedded systems the decisions are usually deterministic but with increasingly complex systems with multiple components being developed the control systems themselves are becoming complicated. The stochastic methods are used heavily to check the robustness and reliability [40]. Due to the complexity of the system and its dependance on multiple components the use of stochastic methods allows extendible modelling of components and abstracts away the details of each component. The extendibility of the stochastic models comes about due to the ease with which the distributions of the components can be incorporated into the system model[82][17][18]. Thus in order to get optimal deterministic control policies for such systems is hard and in some cases may be impossible. One of the methods to get optimal control policies is a class of dynamic programming methods called Markov Decision Processes (MDPs). The application of Continuous-Time Markov Decision Processes (CTMDP) and Generalized Stochastic Petri Nets to do dynamic power management has

been discussed extensively by Pedram et al. in [75] [74]. The use of CTMDP involves modeling the architecture into entities like *Service Providers, Service Requesters* and *Service Queues*. The CTMDP method of modeling provides greater accuracy [76] [74] as compared to *discrete time* methods proposed in [68]. Optimizing the power vs. delay trade off in non stationary load environments has been in part or wholly the motivation of the use of the CTMDP as it is an improvement over the adaptive and predictive methods [10]. An alternative policy generation method using CTMDPs has been suggested by Feinberg [24] [23] the advantages of which are that the mathematical optimality is guaranteed for the solution provided , whereas in [76] the policy may be found but through a much higher computational cost. Another advantage of [24] is that it can be easily applied to problems apart from power management ,whereas the methods in [76] are mostly suited to power management.An example of deterministic methods is the thresholding based control of sleep state of a strong ARM SA-100 processor as discussed in [6]. The method uses a plot of the idle times and active times. The plot shows that the idle times were long enough for the processor to transit into sleep state and return if they were greater than a certain threshold. The threshold was obtained from the plot such that the idle times if occurring beyond the threshold would mostly extend long enough for a valid power down cycle. This threshold is called the break even time because at the threshold the system would just about break even with respect to the power saved in the sleep mode and the power consumed in the transition between modes..

The power management methods mentioned above do not capture all the requirements of an efficient arbitration policy or a policy generation method that comes up with efficient policies. Adapting the methods for arbitration will involve introducing concepts like fairness, mutual exclusion, deadlock prevention and starvation which are critical for designing arbitration policies. The implementation of these concepts has been left to the OS in the existing power management methods [76], but since we are trying to use these methods for bus arbitration the above mentioned requirements should be implicit with the arbiter design and not controlled by the OS.

3.1.3 Adaptive and Predictive Methods

The adaptive and predictive methods are two ways of implementing control. The adaptive methods use events to adapt themselves to the current state of a system whereas the predictive methods can forecast the possible state of a system and adapt themselves to the possible oncoming state[6]. The predictive methods try to predict the time at which a future event would take place and

thus can help optimise the use of resources by shutting down parts that would be expected to be idle for the predicted period of time. The prediction methods have a disadvantage when compared with the adaptive methods which is that the predictive methods need some sort of precharacterisation of the system as well as the workload. The stochastic optimisation methods studied in this report are of the predictive type and depend on queueing models of the architecture and on workloads models as stochastic processes with rates extracted from core graphs of the system.

To get a model for the structure of the architecture is not very hard but a precharacterisation of the workload or traffic is not easy. Efforts have been made in order to characterize the traffic in presence of different traffic intensities and protocols in [102] and [15]. The characterisation of on chip communication while taking into account for diffrent types of failures ,packet drops and latency requirements has been made in [17] and [18]. The performance of the Network on Chip system using stochastic communication was tested and the effect of the data upsets on latency and energy dissipation was studied in [18].

In order to overcome the challenge of complete precharacterisation the adaptive methods have been employed which vary the policies as the system faces new environments.The adaptive methods use some predefined policies in a look up table and interpolate among them if the system load characteristics arent suited to the policies in the look up table[6]. The policies stored in the table are optimal for a certain traffic pattern but due to the dynamic traffic variation they may not be opitmal for all patterns encounterd. The selection of which set of policies to use or interpolate between is judged by analysing the traffic characteristics over a window of time. These methods have been shown to work almost as well as optimal policies computed offline in [6].

3.1.4 Artificial Intelligence Based Methods

Though it is possible to develope optimal control policies with the help of stochastic predictive methods and near optimal policies with the adaptive stochastic policies there is a need to solve for policies offline in both the the above methods. Efforts have been made to use AI based methods to solve constrained based problems [110]. An example has been described in [110] in which the jet propulsion system of a rocket has been modeled with a set of states with control actions dealing with the control of various valves and the stimuli being provided by physical sensors for the controller to gauge the state of the system. These methods perform online searches on graphs that describe the system and make changes to the possible actions that can be

taken in a particular state. In theory these methods have been claimed to have an optimality analogous to the offline methods discussed in the previous subsections [111]. Some of the new concepts in this effort are that they allow for failures of the system which may not have been thought of in the design of the system. Another difference is that there is no offline solving and the system is continuously forced towards different goals. The set of states which can be counted as valid goal states is updated due to the current states and events occuring in the system. Problems with these methods could be explosion of the state space and the need of pruning the space to a set of good goal states ,which has been modeled as an AI problem called Reactive Planning which is an NP Hard problem[110].

3.1.5 Summary

In this section we presented a discussion of the different control methods that have been tried or could be tried for reconfiguration or adaptation in embedded systems. According to us a continuous time,stochastic,online and adaptive method would be an optimal solution to the adaptation control problem and our future work section will deal more with how we plan to obtain such poilicies. The next section deals with the methods we have studied and implemented.These can be classified as continuous time,stochastic,predictive methods.

Chapter 4

Stochastic Modeling of Embedded Systems and CTMDPs for controlling reconfigurable systems

4.1 Introduction

In this chapter we present a stochastic modeling approach for comparing arbitration policies while doing arbiter design. By modeling the architecture as a network of queues, we can observe the average delays experienced by requests, time spent to service a request, transition rates between different states, average queue lengths, and get good estimates for worst case delays and queue lengths. We have applied stochastic modeling to bus arbitration by mapping the components of the architecture to entities like service providers, service requesters and service queues [76].

The policies that have been implemented based on (1)time spent by a request in a buffer (2)number of requests in a buffer and (3)packetisation of the requests. The model has only one ergodic chain which means, all the states communicate with each other and there are no absorbing states. The queue also has finite length and the requests which do not get an empty server at the time they arrive within the system are lost. Once the components have been mapped to one of these entities the operation of the architecture is treated as a Poisson process with interactions between the components modeled as exponential interarrival times with averages maintained constant over large number of state transitions to maintain stationarity [79]. After every transition the queue lengths are examined and recorded and so is the time spent by the system in that state. By using networks of queues we can split the architecture into components that can be examined separately as entities independent of their neighborhood, and thus allowing us to concatenate components into architectures, and then estimate the performance of the arbitration policy for

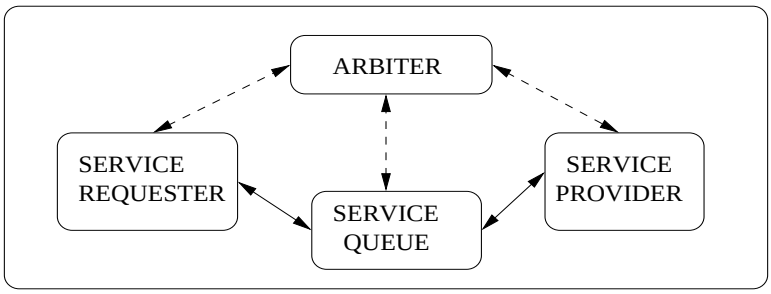

Figure 4.1: **Basic block diagram of a system**

that architecture. The metrics over which policies have been judged are power consumption, delay, loss rate and maximum buffer size. In some scenarios the control will have to be done at different hierarchy levels so there will be dependence among the interarrival times and the control will have to communicate with the control at a higher level. Data is extracted from the simulation model, and used in a set of LP equations [75] [23] to give stochastic policies optimised in terms of power vs. time spent in queue and queue length trade off.

The studied policies are classified as heuristic and stochastic. The stochastic policies can be further classified into deterministic as well as randomised policies.The simulations have been divided into two stages: 1) applying heuristic policies to a queueing model, and extracting transition rates which are used in a CTMDP to find optimal policies for arbitration, and 2) using the policy obtained from the CTMDP on the queueing model and comparing the policies in terms of performance metrics average and worst case queue length, power consumption and loss.

We haven't modeled the delays associated with control signals. The control signals have been assumed to be instantaneous, but some power consumption has been attributed to control signals. The simulation times are short even for medium size systems and small sets of possible topologies for an architecture specification, but could be long for large systems with large number of possible topologies as the delay characteristics will change with topology.

In order to check the feasibility of applying such MDP based methods to arbitration problems we have generated some arbitration policies for a network processor architecture [16] and compared the performance of the policies. We verified the behaviour of these methods in non-Markovian environment and compared them with the classical Markovian environment results. The arbitration policies with mathematical optimality may not be obtainable for some architectures. The explanation of when such situations can occur is beyond

the scope of this thesis, and an effort to go around this problem has been made by adding constraints extracted from the real world system and finding the best sub optimal policy. Policies with mathematical optimality may be obtained but may not be suitable to the real world because the policies are based on models which are incomplete or inaccurrate for example, the model may use a distribution that does not accurately model the traffic,or does not catch the possible congestion or starvation in the system.

4.2 System Modeling Simulation Flow

The proposed design methodology is presented in Figure 4.3. Bus architectures are modeled as continuous time queues, as shown in Figure 4.2(b). The CTMDP models used through the entire flow have only one ergodic chain. This means that all the states communicate with each other, and there are no absorbing states. Queues have finite lengths. The requests, which do not get an empty server at the time they arrive, are lost. The operation of the architecture is treated as a Poisson process. The interactions between components is modeled as exponential interarrival times with averages maintained constant over large number of state transitions. This is needed for maintaining stationarity [79].

The first step of the methodology simulates the system queuing model with an heuristic arbitration policy. Data is extracted from the simulation model, and used next to compute stochastic policies optimized in terms of power consumption under maximum queue length constraints. The heuristic policy looks at the time spent by the request for allowing the clients to serve it. This policy gives unbiased transition rates between states of the system. After every transition, the queue lengths are examined and recorded, and so is the time spent by the system in that state. The traffic has been maintained at stationary exponential rates. This is necessary for the model to represent a CTMC (Continuous Time Markov Chain). The transition rates of the CTMC are extracted from the simulation.

Transition rates are then used to find steady-state probabilities for state-action pairs, i.e. given the system is in a certain state what action will it select. Rates are inputs to a set of linear programming (LP) equations [23] [75]. The LP cost function expresses the power consumption in each transition from one state to the other. The CTMDP, which is an LP using the transition rates of a CTMC and constraints on the queue length as inputs, finds the steady-state probabilities for state-action pairs. Thus, the policy that we talk about is a set of probabilities associated with choosing certain actions when in a given state.

Policies are optimal in the sense that they find the transitions that consume the least power while keeping the queue lengths within fixed constraints.

After obtaining the probabilities for state-action pairs, the randomized stochastic policy [76] and the KSwitching strategy [24] are simulated. The constraints to the LP equations are then changed, and a deterministic policy is obtained. The deterministic policy is then simulated on the queuing model. Heuristic policies, such as time spent, time + length, and round robin policies, are then also implemented. The results of the simulation for all (heuristic and stochastic) policies are collected and plotted. In our experiments, simulations were carried out in Matlab 6.5 [57]. The queuing models and simulation were done based on [48] and [54]. The performance metrics over which policies have been judged are power consumption, data loss rate, average and maximum buffer size.

After arbitration policy selection, the methodology redistributes the buffer space to reduce data loss. The selected arbitration policy is simulated on the queuing model with a constant amount of buffer space allotted to a particular bus-processor pair. Data loss is recorded. The loss and buffer size used are utilized as costs in the CTMDP-based redistribution of the buffer space. Then, the policy is re-simulated on the architecture with customized buffer space distribution, as shown in Figure 4.3. The results of the data loss as well as the new distribution are plotted, and compared with other buffer space distribution policies, like time-out policy.

By using networks of queues, we can split the architecture into components that can be examined separately as entities independent of their neighborhood. This allowed us to concatenate components into architectures, and then estimate the performance of the arbitration policy for that architecture. In some scenarios the control will have to be done at different hierarchy levels. So, there will be dependence among the interarrival times, and the control will have to communicate with the control at a higher level.

We haven't modeled the delays associated with control signals. The control signals have been assumed to be instantaneous, but some power consumption has been attributed to them. Simulation times are short even for medium size systems, but could be long for systems with large number of possible topologies as delay attributes change with topology.The bus arbitration policies that have been implemented based on (1) time spent by a request in a buffer, (2) number of requests in a buffer, and (3) packetization of the requests.

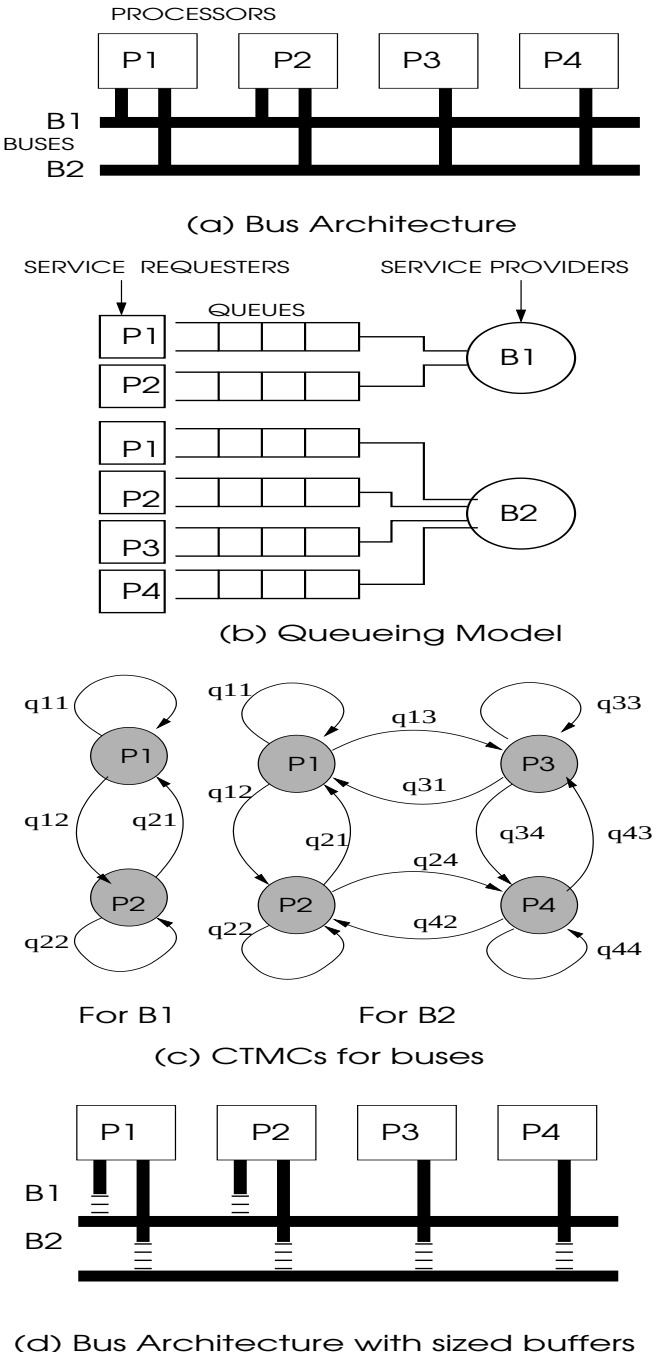

Figure 4.2: **Design Steps**

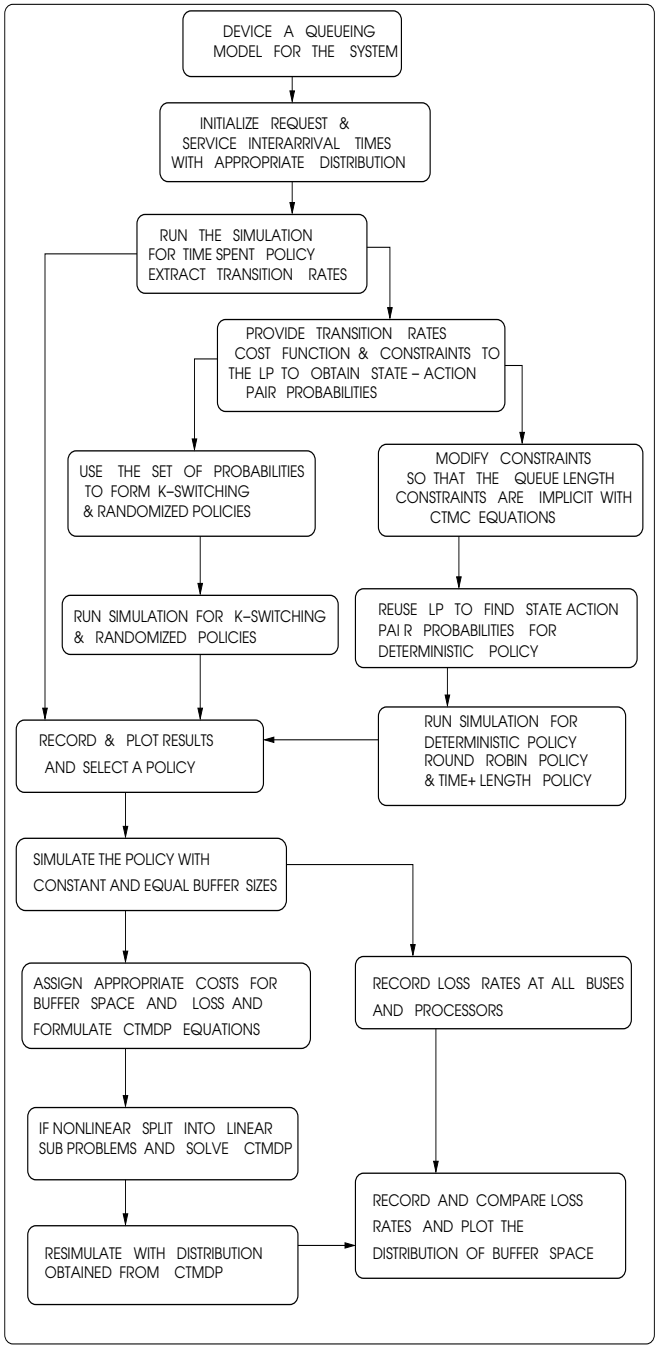

Figure 4.3: **Simulation flow**

4.3 Discussion about Policies Implemented

Once the policy has been obtained it is implemented over the queueing model .The following are the policies which have been implemented:1)Round Robin,2)Time+Length policy,3) Randomised stochastic policy [76], 4)Deterministic stochastic policy, and 5)K-switching strategy [24]. The policies that have been compared are by no means an exhaustive set of available arbitration policies, but we have tried to present a set that would enable us to show a comparison between the basic flavours of policies available as well as some very recent stochastic policies, like K-Switching strategy.

4.3.1 Heuristic Policies

Round Robin

The server is accessible in turn by all processors irrespective of how long their requests have waited in the queue, the queue lengths, or the power consumed by the sequence of transitions followed in the round robin. The drawback of the roundrobin policy is that the order of accessing the resource is fixed and requests have to wait for their turn, thus increasing the time spent in buffers and the required buffer lengths.

Time+Length Policy

The Time+Length policy looks at the queue lengths in the buffers, and selects the request waiting at the beginning of the longest queue to be serviced by the bus. In case there are two or more longest queues then it selects the request which has waited longest. In case the time spent by the requests is also the same then it will randomly pick one of them. This policy manages the queue lengths and the time spent in buffers well, but is blind to aspects like regulating power consumption and deadlock prevention.

4.3.2 Stochastic Policies

These policies have been obtained through a set of LP equations, which were given inputs regarding constraints and transition rates by extracting them from a simulation of the queueing model of the system as shown in Figure 4.3.

The state transition probabilities are extracted and then used to find the steady state probabilities of the embedded Markov Chains. These probabilities have been used in the simulation to verify the correctness of the simulation as well as estimate valid constraints and rewards when the LP is used. The LP

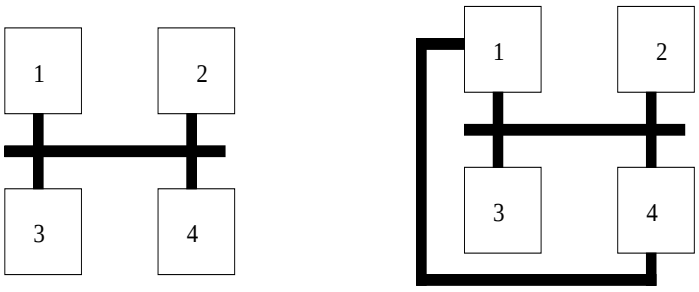

Figure 4.4: **Topology with and without redundant paths**

takes the transition rates and constraints and the reward function as inputs, and gives a set of probabilities for the state action pairs. These are used by the arbiter to randomly assign the bus to a processor.

Randomised Stochastic Policy

This is an ideal case and results in an optimal policy [24], and has been used as a yardstick for the rest of the policies. For the LP to return a randomised stochastic policy all the constraints for all actions should push the system toward an optimal solution. In case for a particular state the constraint isn't valid, and does not effect the system then the policy will have a deterministic action associated with that particuar state.

Deterministic Stochastic Policy

In a Deterministic stochastic policy though the approach to device the policy is to use a CTMDP, the solution provided by the LPs will choose only one action in every state with probability one. Depending on the constraints that are given to the LP, the output solution it provides could be, and and is more likely to be a deterministic policy [68]. These deterministic policies aren't optimal and finding an optimal policy among the deterministic policies is an NP hard problem. The LP gives a deterministic policy when the constraints imposed on the queue lengths are implicit with in the equality constraints imposed by the CTMC. In case the constraints further reduce the space, and aren't implicit then the policy turns out to be a randomised policy.

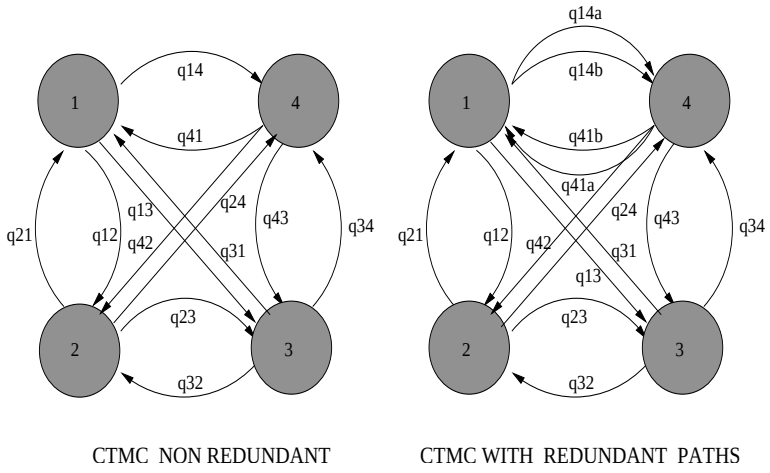

Figure 4.5: **CTMC for topologies shown above**

K-Switching Strategy

Another approach is to use a piecewise constant policy, i.e over a certain period of time it will select only one action in a given state. Though this policy is deterministic over intervals of time, it has to assign every action in a state some amount of time. This time interval over which it implements exactly one action is variable, and depends on the state action pair probabilities which are obtained from the LP. Theoretically this policy defines a similar semi Markov chain over the space as the randomised stochastic policy, and thus the rewards that it will earn will be the same as that of the optimal randomised stochastic policy[23] [24]. Thus this policy has been proposed instead of the finding an optimal policy among deterministic policies, which aren't optimal when compared to the randomised policy.

4.4 Mathematical Modelling

In case we know the timing characteristics of the service requests it may be possible to avoid lengthy simulation by using analytical methods to find the the required buffer sizes and arbitration policies but these would fail to explore all the possible scenarios and would give only average or worst case scenarios. Apart from that the numbers produced by the analytical methods can be compared with the results of the simulation to validate the simulation results. we ran several simulations to study M/M/1, M/D/1, M/G/1, M/M/1/K, M/D/1/K and M/G/1/K queues in order to compare their characteristics.

Some of the results of have been displayed in Appendix II. The infinte buffer space queues were approximated by finite queues with a buffer size of two million.

4.4.1 Finite number of Customers

We initially assumed we have a finite number of customers M each with an *arriving* parameter of λ. We also have m number of servers with *service* parameter μ. The system also has a finite amount of storage room such that the total number of customers in the system (queueing plus those in service) is no more than K. We assume M $\leq$ K $\leq$ m,i.e customers arriving to find K customers already in the system are lost [48].

(λ)=arrival rates, (μ)=service rates
M=Number of customers, k= customers in sytem, m=number of servers

$$\lambda_k = \begin{cases} \lambda(M-k) & if\ 0 \leq M \leq k-1, \\ 0 & otherwise \end{cases}$$

$$\mu_k = \begin{cases} (\mu)k & if\ 0 < k < m, \\ (\mu)m & if\ k > m \end{cases}$$

Case1: Multiple Servers
There are two cases that have to be considered 1) if number of customers in queue $\leq$ number of servers and 2) if number of customers in queue $\geq$ number of servers

$0 \leq k \leq m-1$

$$p_k = p_0(\prod_{i=0}^{k-1})\frac{\lambda(M-i)}{(i+1)\mu} \tag{4.1}$$

The probability p_k is the probability of there being K customers with in the system. The model has customers with arrival service rates which also vary with time but the changes in the arrival rate that have been dealt with in these experiments are sudden jumps which occur rarely and the rates have been maintained constant over large stretches of time in case there are multiple service providers their rates of service could be differ from each other but are maintained constant throughout the experiment, thus the equation for the queue being in state K can be written as:

$$p_k = p_0(\prod_{i=0}^{k-1})\frac{\sum_{i=1}^{k-1}(\lambda_{M-i+1})}{\sum_{j=1}^{i+1}(\mu_j)} \tag{4.2}$$

$m \leq k \leq K$

$$p_k = p_0(\prod_{i=0}^{m-1})\frac{\lambda(M-i)}{(i+1)\mu}(\prod_{i=m}^{k-1})\frac{\lambda(M-i)}{m\mu} \tag{4.3}$$

as in previous case

$$p_k = p_0(\prod_{i=0}^{m-1})\frac{\sum_{i=1}^{k-1}(\lambda_{M-i+1})}{\sum_{j=1}^{i+1}(\mu_j)}(\prod_{i=m}^{k-1})\frac{\sum_{i=m}^{k-1}(\lambda_{M-i+1})}{m\mu} \tag{4.4}$$

Case2: Single Server In case there is just one server the cases reduce to one case.

$$p_k = p_0(\prod_{i=0}^{k-1})\frac{\lambda(M-i)}{(i+1)\mu} \tag{4.5}$$

but the model has customers with different service rates

$$p_k = p_0(\prod_{i=0}^{k-1})\frac{\sum_{i=1}^{k-1}(\lambda_{m-i+1})}{\sum_{j=1}^{i+1}(\mu_j)} \tag{4.6}$$

We may calculate the expected number of requests in the system from

$$N = \sum_{k=0}^{M} kp_k \tag{4.7}$$

The arbitration policies will be looking at the queue lengths and the average time spent to make decisions. The power consumption could be related to the time spent by a request in the system. As long as the resource is idle we could say that it isn't consuming any power. Thus a greedy policy which shuts off the service provider as soon as the service is done would be the best if trying to save power is the only goal.

In many cases the the interarrival times may not be exponential and this will cause the loss of Markovian property but this problem has been elegantly dealt with in [75] by using the *stage method* in which the non-exponential service requests could be handled by using series or parallel connections of exponential servers.The method allows analysis of non stationary service requests as the Markovian property will be preserved in the *Erlangian distribution* resulting from the combination of exponential servers.The timing characteristics of the data may be non exponential though [75][74].

4.4.2 Finite Buffer Space

We found that the finite population model does not fit the bus architectures requirements well. The finite population model requires a finite number of

customers cycling to the system but in reality there can be any number of customers but they have to be appearing from a finite number of sources or in our case processors. So the buses can talk only to a finite number of processors but the processors can request the bus as much as the input data or the processing taking place within them would require. To model this we have used an M/M/1/K queueing model.

It has been assumed that all processors have poisson process for generating requests. The processors make requests to different buses depending upon which other processor they must talk to. The poission process is further split into poision processes of lower rates depending on how many buses the processor talks too. From the bus side looking into the queue the bus recieves requests from various processors. We let the rate at which the bus gets requested be (λ). The following equations for a M/M/1/K system have been obtained from [48].

$$\lambda_k = \begin{cases} \lambda & if\ k \leq K, \\ 0 & otherwise \end{cases}$$

$$\mu_k = \mu \quad k = 1, 2, 3.....K$$

The probability of there being k customers in teh queue is given by p_k

$$p_k = p_0 \prod_{i=0}^{k-1} \frac{\lambda}{\mu} \quad k \leq K \tag{4.8}$$

which is the same as

$$p_k = p_0 \left(\frac{\lambda}{\mu}\right)^k \quad k \leq K \tag{4.9}$$

and

$$p_k = 0 \quad k \geq K \tag{4.10}$$

On solving for p_o and p_k we get

$$p_0 = \frac{1 - \lambda/\mu}{1 - (\lambda/\mu)^K + 1} \tag{4.11}$$

$$p_k = \begin{cases} \frac{1-\lambda/\mu}{1-(\lambda/\mu)^K+1} \left(\frac{\lambda}{\mu}\right)^k & 0 \leq k \leq K, \\ 0 & otherwise \end{cases} \tag{4.12}$$

We may calculate the expected number of requests in the system from

$$N = \sum_{k=0}^{K} k p_k \tag{4.13}$$

The finite buffer space has been used as a constraint while designing the control policies with the CTMDP methods. Certain observations which would be useful for our methodology would be as follows.

- For finite queues the loss rate can be predicted in case the server utilization, and service rates are known.
- The waiting time in the queue tends to the product of the average service time with the buffer size as the server utilization increases.
- The dependence of the loss rate on the service time distribution. For the same server utilisation level for finite queues the loss rates were found to be different fordifferent service rate distributions

4.5 Transition from Queueing Model to CTMDP based optimisation Problem

The CTMC is defined by a transition rate matrix. The transition rate matrix is modified to form the transition rate matrix of a CTMDP by incorporating the concept of actions. The transitions between any two states could occur with different rates depending on the the action selected. In our case the action would be a choice between redundant paths for transmitting data. The physical significance of this redundant path will be that the data transfer will have different latency as well as power consumption depending on the path. The redundant paths are made available to balance the communication load across the buses in order to have minimal conflict as well as lower bus speed requirements [99] [100]. The arbitration policies will have to find efficient distribution of the load among the buses while managing the queue lengths and power consumed by IP cores. The IP cores will consume power in their steady state and the data transfer will have different rates. These facts are used in the CTMDP to assign different rewards to the steady state power consumption and the different paths.

The transition rate matrix of the CTMDP can be used as an input to an LP, which find an optimal set of state action probabilities to optimise the power consumption in the architecture while minimising the queue lengths and the time spent by requests in the queues. The constraints needed for the LP can be obtained by simulating an unbiased arbitration policy on the queueing model and recording the average and worst case queue lengths, the average and the max time spent by request in the queue. While fixing the constraints

consideration has been given to concepts important to any arbiter like fairness, prevention of deadlock and starvation.

4.5.1 Problem Formulation for Bus Arbitration

This section gives a brief yet pertinent explanation of Continuous Time Markov Decision Processes (CTMDP). An in depth study of Average Reward Constrained Continuous-Time Markov Decision Processes can be found in [23][24][25]. A Continuous-Time Markov Decision Process is the set

$$\{I,A,A(.),q,K,r\} \ ,$$

I is a finite state space, A is a finite action set, and $A(i)$ is a set of actions available at state $i \in I$. $q(i,j,a)$ is a transition rate from state $i \in I$ to state $j \in J$, if action $a \in A(i)$ is selected. $q(i,j,a) \ \geq 0$ for $i \neq j$, and $\sum_{j \in I} q(i,j,a) = 0$ for all $i \in I$. $K \ = 0,1....$ is the number of criteria, $r_k(i,a)$ is a reward rate for criterion $k \ = 0,..,K$, if action a is selected in state i.

In the thesis, the concept of state is defined as the processors, which is to be assigned to buses at a given instant. Every bus is associated with a Markov Chain. The chain is in state i, if processor i is putting data on that bus. Action A is to make a choice between a set of redundant paths between the IP cores. Since paths have different lengths, and data on them suffers different delays, rates $q(i,j,a)$ also differ. Figure 4.5 shows the CTMCs for the bus architectures without and with redundant paths shown in Figure 4.4. The differing rates are chosen with differing probabilities as a consequence of the state-action pair probabilities generated by the LP. $A(i)$ are the set of available actions in a particular state. These vary from one to how many ever redundant paths are available from that state to the set of possible next states. The reward function, which is maximized by the LP, has one performance quantity over which optimization is taking place, such as power consumption.

The transition rates when normalized give the state transition probabilities for the Markov Chain, as if the transition rate is infinite for the CTMDP for a particular transition occurring due to a particular action. One way of looking at it is as if the actions are defining sets of Markov Chains on the space. The state transition probabilities were computed, and then used to find the steady state probabilities of the embedded Markov Chains. These probabilities have been used in the simulation to verify the correctness of the simulation, as well as estimate valid constraints and rewards when the LP is used. The LP takes the transition rates and constraints, and the reward function as inputs, and gives a set of probabilities for the state action pairs. Computed probabilities were then used by the arbiter to assign a bus to a processor. There is an issue

with the action being the next processor which sort of merges the concept of action with the concept of next state. Thus, the action may have to be isolated in a physical sense from the next state, i.e. the action could be redundancy in paths while making a transition, different sets of priorities or traffic scenarios, which will define different transition rates on the queuing model. In the thesis we have used redundant paths among the processors as the actions.

For a system in which multiple processors can simultaneously access multiple buses, each combination of states in the Markov Chains (MCs) for the buses could define a state for the overall system. For example in Figure 4.2(a), a possible state of the system is the set {(*P4*, *B1*), (*P1*, *B2*)}, where processor *P4* accesses bus *B1*, and processor *P1* communicates using bus *B2*. The two pairs are states in the MCs for the two buses. Figure 4.8 shows how the state is represented. The figures shows at time t processor 1 has possesion of bus A and bus B. At time *t+1* processor 1 looses bus B to processor 3. The state of the the period from time instant t till time instant *t+1* would have been X(1,a)X(1,b) and at time instant *t+1* the state would change to X(1,a)X(3,b). The number of such processor bus pairs which are needed to define the system state would be equal to the number of buses. Considering the states of the overall system would be useful, if the access of certain processors to buses has to be coordinated, e.g., specific data communications must be synchronized. However, the thesis does not consider such requirements, and therefore it did not utilize the state of the overall system for defining CTMDP models. In addition, this modeling procedure would result in models with a very large number of states.

Constraints prevent starvation and deadlock, and provide fair allocation of resources. By applying bounds on the space in which the LP searches for a solution, we can limit the space to regions where it can't give solutions that cause starvation. The deadlock occurrence may not be prevented completely by the constraints and bounds placed on the LP, but it can be reduced. There are four conditions that have to occur in order to have deadlock in a system [91]:

- Mutual exclusion condition: each shared resource is allocated to exactly one process at a time.
- Hold and wait condition: processes currently holding resources can ask for new resources.
- No preemption condition: resources once allocated can't be forcefully taken away, they have to be released by the process.
- Circular wait condition: there is a chain of two or more processes, each waiting for a resource held by the other.

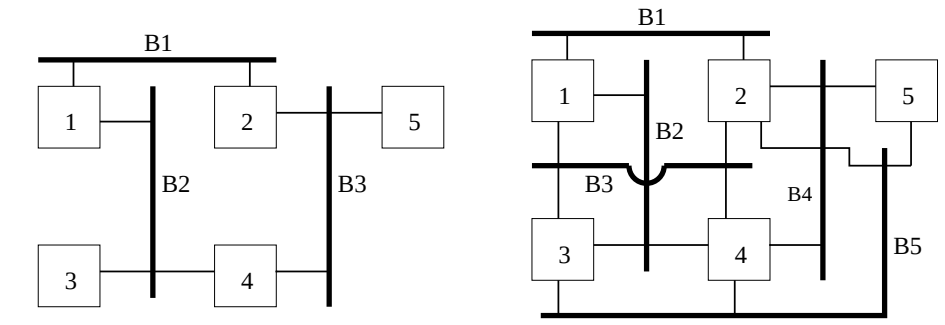

Figure 4.6: **Bus architectures for Examples 1, 2, and 3.**

We reduced deadlock by attacking the last condition. This has been done by exploring the transition rate matrix for possible cycles, where the transitions are high among processors that share more than one resource. The cost function was modified, such that these cases are converted to cases where one resource is favored in one of the states, and the second in the other, thus reducing the possibility of deadlock.

Different actions are associated with different reward rates $r_k(i,a)$, due to the different rates available to the Markov Chain when a separate action is chosen. These rewards and the transitions rates with constraints obtained from the first simulation run are given to a set of equations shown below. The reward (cost) function is made up of the possible expected rewards that can be obtained by choosing an action, as well as the rewards that are earned while in that particular state. Constraints are on the queue lengths that occur while the buses serve the processors.

$$Maximize \sum_{i \in I} \sum_{a \in A(i)} r_o(i,a) x_{i,a}$$

Such That

$$\sum_{a \in A(j)} q(j,a) x_{j,a} - \sum_{i \in I} \sum_{a \in A(i)} q(i,j,a) x_{i,a} = 0, j \in I,$$

$$\sum_{i \in I} \sum_{a \in A(i)} r_k(i,a) x_{i,a} \geq C_k, k = 1, ..., K,$$

$$\sum_{i \in I} \sum_{a \in A(i)} x_{i,a} = 1,$$

$$x_{i,a} \geq 0, i \in I, a \in A(i),$$

Transition rates meet the constraint *q(i,a) = -q(i,i,a)* [24] and as shown in Figure 4.9. The figure shows a processor and how the rates of the continuous

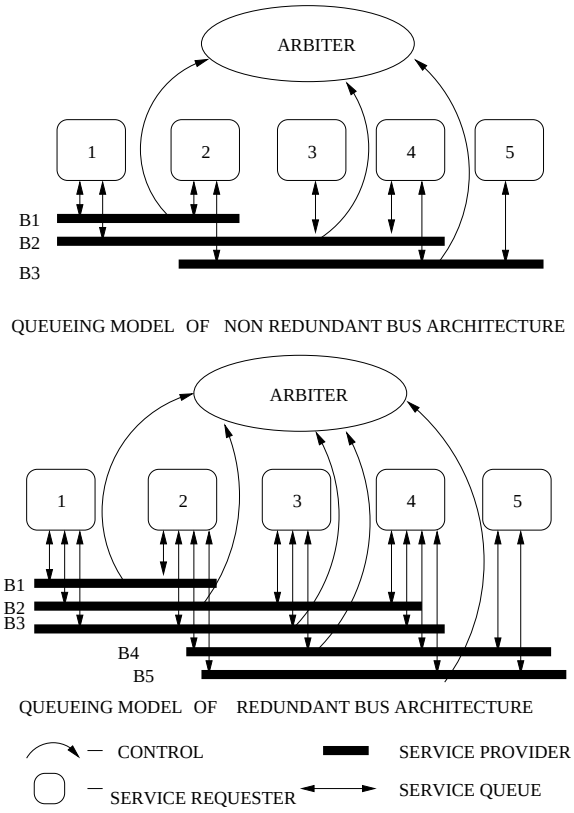

Figure 4.7: **Queuing models for Figure 4.6**

time markov process relate to the state transitions. The rate at which the processor regains control of the bus is used as a measure of the rate of how long the bus is in possesion of that particular processor. The second relation which is brough about by the equality constraints of the markov chain is depicted as the sum of the rates of the bus being provided to all processors j from processor i is equal to the the rate at which the bus is provided to processor i from processor i itself. The set of LP equations consist of a reward function, which has to be maximized, and a set of equality constraints derived from the steady-state equations of the Markov Chains and a concept called uniformization. Uniformization is a method to reduce the Continuous Time Markov Process to a Discrete Time Markov process, which is equivalent to the continuous time process in terms of the rewards or reward rates associated with the states [24][25]. There is a set of inequality constraints obtained from simulations, and bounds on the space that the LP can find a solution in. The output of the LP is a set of state-action pair probabilities, which are the long run probabilities of choosing a certain action, if the system is in a given state.

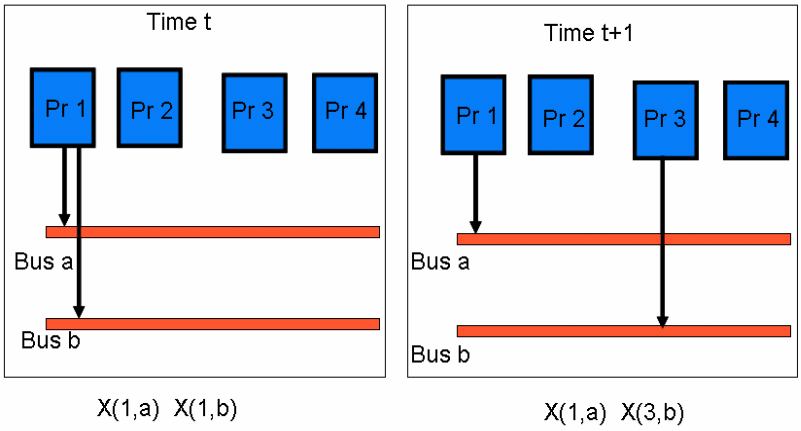

Figure 4.8: **Representation of the system state**

Example 1: Given below are the equations for bus *B2* of the architecture with redundant buses between processors shown in Figure 4.6. Its queuing model is in Figure 4.7.

Maximize

$$r_{1,b2}x_{1,b2} + r_{3,b2}x_{3,b2} + r_{4,b2}x_{4,b2} \tag{4.14}$$

Equality constraints are

$$q_{1,b2}x_{1,b2} = q_{1,1,b2}x_{1,b2} + q_{3,1,b2}x_{3,b2} + q_{4,1,b2}x_{4,b2} \tag{4.15}$$

$$q_{3,b2}x_{3,b2} = q_{3,3,b2}x_{3,b2} + q_{1,3,b2}x_{1,b2} + q_{4,3,b2}x_{4,b2} \tag{4.16}$$

$$q_{4,b2}x_{4,b2} = q_{4,4,b2}x_{4,b2} + q_{1,4,b2}x_{1,b2} + q_{4,3,b2}x_{4,b2} \tag{4.17}$$

Another modeling alternative would be to define MC states that also consider the state of queues, e.g., instantaneous queue length. This modeling would be more accurate, and offer better control over queue attributes, like maximum queue length. However, the models suffer from an extremely large state space. The possible states that a queue could take are from zero occupancy to the maximum possible queue size. This is much larger than the possible states in the current formulation. The present formulation uses an average estimate of the queue length. For any FIFO queue, the average length

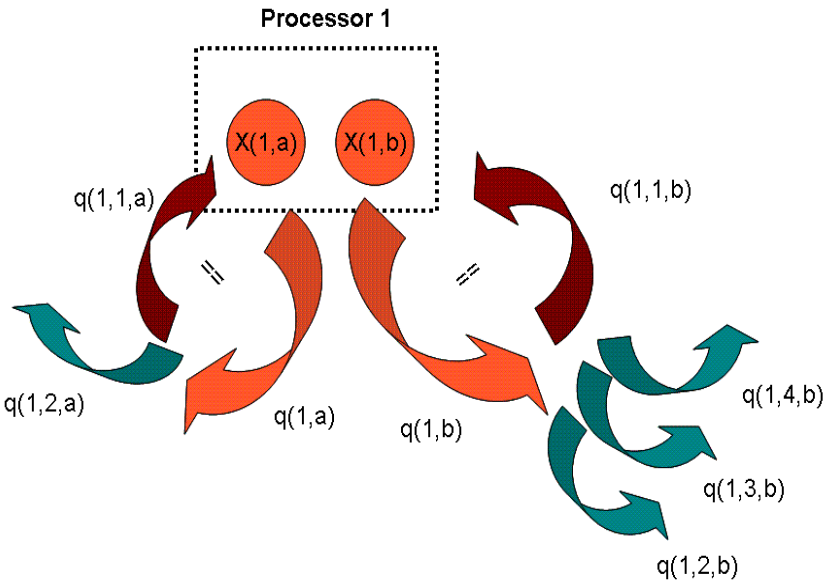

Figure 4.9: **Relation between the continuous rate quantities**

of queues is directly proportional to the rates at which requests fill in the queue [79]. Queue lengths are inversely proportional to service rates [79], but all queues will see the same service rate. Hence, the dependence of queue lengths is solely on the arrival rates.

4.5.2 Randomized and Deterministic Stochastic Policies

The policy allocates the appropriate bus to a processor as per the state-action pair probability of that processor needing a bus and getting allotted a certain bus.

$$\varphi(\overline{a}|i) = \frac{q(i,\overline{a})x_{i,\overline{a}}}{\sum_{a\in A(i)} q(i,a)x_{i,a}} \text{ if } \sum_{a\in A(i)} q(i,a)x_{i,a} > 0,$$

otherwise

$$\varphi(\overline{a}|i) = I(\overline{a} = a), \textit{where } a \textit{ is an arbitrary element of } A(i)$$

This set of probabilities is directly used to implement a randomized stochastic arbitration policy, which chooses the buses for processors in a random fashion. Though random, the probabilities are such that they optimize power consumption. Since every processor doesn't talk to every bus, the number of actions, and the number of possible actions available at every state are different. They are defined by the set $A_x = a \in A(i) : x_{i,a} > 0$, for all valid actions in a state.

In case the action chosen in the above policy is the same each time the system is in a given state, the resulting policy is called a *deterministic stochastic policy*. This situation occurs if the constraints do not force the LP to use the actions with higher cost. For example, the action in a given state with the maximum reward can be chosen every time without violating any of the constraints.

Example 2: For a randomized stochastic policy φ, the probability of using bus *B2* for traffic from processor 1 is

$$\varphi(b2|1_{b2}) =$$

$$\frac{q(1_{b2}, b2)x_{1_{b2},b2}}{q(1_{b2}, b1)x_{1,b1} + q(1_{b2}, b2)x_{1_{b2},b2} + q(1_{b2}, b3)x_{1_{b2},b3}}$$

The state has been defined with a subscript of *b2* to signify that this equation is for bus *B2*.

4.5.3 KSwitching Strategy

The alternative policy generation method proposed in [24] is as follows. The probabilities obtained from the LP are converted to lengths of time over which the system performs a particular action while in a particular state. Thus, the processors get allotted buses for periods of time, which depend upon the state-action pair probabilities. After fixing a certain ordering of the actions, the policy cycles through the whole set actions. In our methodology, this means that given a processor has to talk to another, then in case there are redundant paths available, it uses the redundant paths for time lengths in proportion with the probabilities given by the LP.

$$sn_l(i, x) = \frac{\ln(1 - q(i, a(i, l)))x_{i,a(i,l)}}{\sum_{m=l}^{n(i,x)} q(i, a(i, m))x_{i,a(i,m)}} \tag{4.18}$$

$$s_l(i, x) = -\frac{sn_l(i, x)}{q(i, a(i, l))} \tag{4.19}$$

$$\psi(i,t) = a(i,l), if A_x(i) \neq a, and\ S_{l-1}(i,x) \leq t < S_l(i,x)$$

$$\text{where } a \text{ is an arbitrary element of} A(i), if A_x(i) = \phi$$

s_l is the time length between two successive epochs, hence $S_l(i,x) = S_{l-1}(i,x) + s_l(i,x)$. The above equation signifies that between epochs $S_{l-1}(i,x)$ and $S_l(i,x)$ the system always chooses action $a(i,l)$ given it is in state i. Hence, the KSwitching strategy has the same average rewards as the randomized stochastic policy. The reason for using KSwitching strategy is that LP may return a deterministic stochastic policy, which isn't optimal. In case the LP returns only deterministic policies then it is an NP-hard problem to find an optimal policy among the deterministic policies. In other words a stationary optimal policy may not exist for a certain problem. In order to avoid this problem, we used KSwitching policy, which is a piecewise linear approach towards solving the problem, and gives us the best sub-optimal policy. [24] offers more insight into this problem.

Example 3: In Figure 4.7, the length of time that bus *B2* is in control of processor 1 under KSwitching policy is proportional to $s_l(1_{b2}, b2)$, which is given by:

$$sn_l(1_{b2}, b2) = \frac{\ln(1 - q(1_{b2}, a(1_{b2}, l)))x_{1_{b2},a(1_{b2},l)}}{\sum_{m=l}^{n(1_{b2},b2)} q(1_{b2}, a(1_{b2}, m))x_{1_{b2},a(1_{b2},m)})}$$

$$s_l(1_{b2}, b2) = -\frac{sn_l(1_{b2}, b2)}{q(1_{b2}, a(1_{b2}, l))}$$

Where l is an index for the epochs. m is an index of all possible actions available at a state in a certain inter-epoch period. It is necessary to tag the terms with the discrete index l because the set of actions available at the state could be different at different epochs. This is due to the derivation of this policy from Randomized Markov Policy, which depends on the number of jumps and the current state to decide on which action to select [23] [24].

4.6 Buffer Insertion Method

The optimal buffer sizing problem was formulated in a CTMDP framework as follows. Similar to CTMDPs for bus arbiter modeling, states are defined by processors accessing a bus in the system. An action is to give a certain processor control over the bus. Rates *q(i, a)* are the request rates of the processors. Rates *q(i, j, a)* are the rates of processor i giving up bus a for

another processor *j*. Traffic rates *q(i,j,a)* and *q(i,a)* were obtained from simulating the system queuing model with time + length policy and equal amount of buffer space allotted to every processor-bus pair (see the design flow in Figure 4.3). Future work could improve the estimation of the rates by simulating the models for other policies also. However, the challenge is to do an educated guess about the policy, which offers the best predictions. Exhaustively considering all policies results in very long simulation times. Rewards $r_o(i,a)$ are the average loss rates obtained from simulation. Coefficients $r_k(i,a)$ are the average queue lengths encountered in simulation. Different actions were associated with different reward rates due to the different rates available to the Markov Chain when a separate action is chosen. These rewards as well as the transitions rates were used to set up the CTMDP model shown below.

The buffer insertion problem was formulated as a negative dynamic programming problem [25], as rewards are negative quantities. The cost function has to be maximized The value function or the possible value taken by the cost function is optimal, if it has the maximum value. (since costs are negative). we maximized them, had they been positive quantities, we would have minimized the costs. We used LP-based methods for solving the CTMDP. CTMDP models for buffer insertion involve five kinds of relationships: (1) equality constraints, (2) inequality constraints, (3) cost function, (4) lower bounds, and (5) upper bounds. The constraints in the set of equations physically relate to the finite buffer space available for a bus. Inequality constraints ensure that the used buffer space is less than a certain fixed quantity. The arrival rates used in the simulation were obtained from the core graph in [99]. The service rates were related to the bus bandwidth. The reward (cost) function is made up of the expected rewards that are obtained by choosing an action, and the rewards that are earned while in that particular state. The total reward was maximized. Constraints are on the queue lengths that occur while the buses serve the processors.

$$Maximize \sum_{i \in I} \sum_{a \in A(i)} r_o(i,a) x_{i,a} \tag{4.20}$$

Such That

$$\sum_{a \in A(j)} q(j,a) x_{j,a} - \sum_{i \in I} \sum_{a \in A(i)} q(i,j,a) x_{i,a} = 0, j \in I, \tag{4.21}$$

Similar to the modeling in Section III, the CTMDP model has a set of equality constraints derived from the steady-state equations of the Markov Chains, and the uniformization concept. Uniformization reduces a Continuous Time Markov Process to a Discrete Time Markov Process, so that the two are equivalent in terms of the rewards or reward rates associated with the states [24][25].

Inequality constraints ensure that the used buffer space is less than a certain fixed quantity.

$$\sum_{i \in I} \sum_{a \in A(i)} r_k(i,a) x_{i,a} \leq C_k, k = 1, ..., K, \tag{4.22}$$

A set of inequality constraints (as shown below), had been obtained from the simulations exploration loop, and bounds were placed on the space the LP can find a solution in. In addition,

$$\sum_{i \in I} \sum_{a \in A(i)} x_{i,a} = 1, \tag{4.23}$$

$$x_{i,a} \geq 0, i \in I, a \in A(i), \tag{4.24}$$

Transition rates meet the constraint *q(i,a) = - q(i,i,a)* [24]. The optimal distribution of this space, as well as insertion of extra buffer space is obtained from the solution of this model. The solution is a set of state-action pair probabilities, which are then translated into physical quantities for the buffer space. The translation uses the KSwitching policy [24] over which the system performs a particular action while in a particular state. The KSwitching policy has been explained in Equations 4.18 and 4.19 of the previous subsection. Thus, processors get allotted buffer space depending upon the state-action pair probabilities, i.e. which bus it is talking to. In our system this means that the buffer space gets divided into a certain ratio in proportion with the probabilities found from the model solving.

Up to this point, the methodology considered only bus architectures that handle only communications between processors. However, in Figure 4.10, Architecture 2 has buses *b, f*, and *g* that are connected to each other through bridges. The communication between processors 2, 3, and 5 involves insertion of buffers, and requires a controller to take into account the traffic from all three processors while making arbitration decisions for any of these three buses. One of the problems with designing such an arbiter is that it requires solving quadratic equations due to the interaction between two buses. In case the buses talk to each other through bridges, the equality constraints and the cost function have quadratic terms. An equation may have more than one quadratic term.

Example 4: To illustrate what kind of equations exist in case buses talk to buses, we presented below the equations for bus *b* in Architecture 1 and Architecture 2, as shown in Figure 4.10. The equations for bus *b* in Architecture 1 are:

Maximize

$$r_{2,b} x_{2,b} + r_{3,b} x_{3,b} \tag{4.25}$$

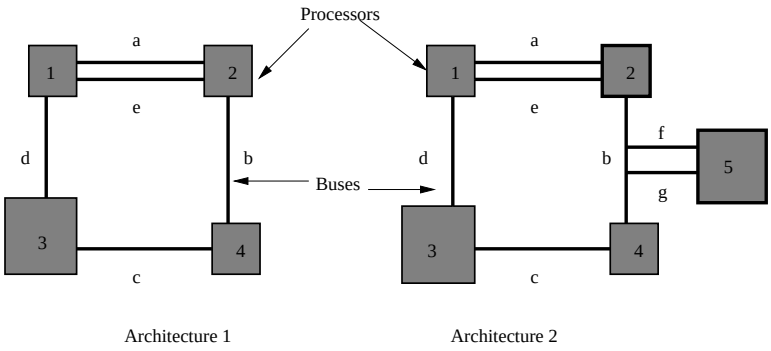

Figure 4.10: **Bus architecture which need splitting**

Equality constraints

$$q_{2,b}x_{2,b} = q_{2,2,b}x_{2,b} + q_{3,2,b}x_{3,b} \tag{4.26}$$

$$q_{3,b}x_{3,b} = q_{3,3,b}x_{3,b} + q_{2,3,b}x_{2,b} \tag{4.27}$$

For bus b in Architecture 2, the equations with quadratic terms are:
Maximize

$$r_{2,b}x_{2,b} + r_{3,b}x_{3,b} + r_{5,f}x_{5,f}x_{5,b} + r_{5,f}x_{5,g}x_{5,b} \tag{4.28}$$

Equality constraints

$$\begin{aligned} q_{2,b}x_{2,b} = {} & q_{2,2,b}x_{2,b} + q_{3,2,b}x_{3,b} + q_{5,2,b}x_{5,f}x_{5,b} \\ & + q_{5,2,b}x_{5,g}x_{5,b} \end{aligned} \tag{4.29}$$

$$\begin{aligned} q_{3,b}x_{3,b} = {} & q_{3,3,b}x_{3,b} + q_{2,3,b}x_{2,b} + q_{5,3,b}x_{5,f}x_{5,b} \\ & + q_{5,3,b}x_{5,g}x_{5,b} \end{aligned} \tag{4.30}$$

$$\begin{aligned} q_{5,b}x_{5,f} + q_{5,b}x_{5,g} = {} & q_{3,5,b}x_{3,b} + q_{2,5,b}x_{2,b} \\ & + q_{5,5,b}x_{5,f}x_{5,b} + q_{5,5,b}x_{5,g}x_{5,b} \end{aligned} \tag{4.31}$$

The solution we propose for this problem is to split the bus architecture into a set of independent linear systems separated from each other by buffers. The set of equations for each one of the linear modules is solved separately. The fashion in which the system is split is shown in Figure 4.11. The system has been split into four subsystems, as shown in Figures 4.12 and 4.13. Each of the four subsystems has a set of linear equations associated with it. In order to find the optimum for the entire system, all equations are solved together, and not sequentially for each subsystem. In Figure 4.12, buses *b*, *f*, and *g* for subsystem 1 were initially communicating. After the split, bus *b* becomes

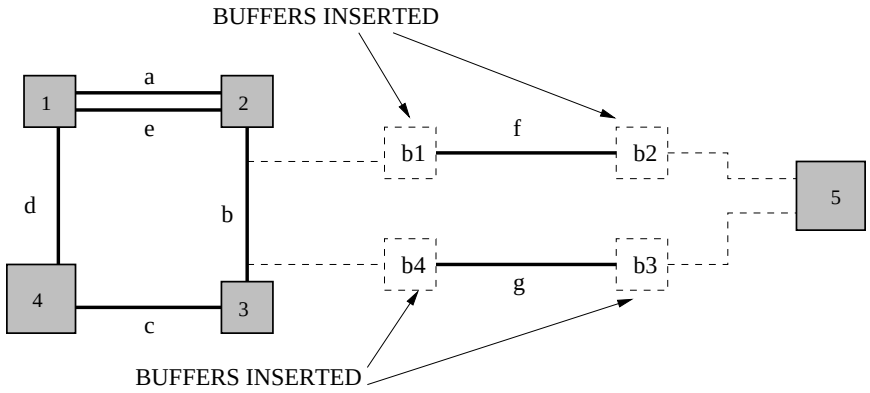

Figure 4.11: **Bus architecture splitting and buffer insertion**

a shared resource between buffer *b1*, buffer *b2*, and processors 2 and 3, thus isolating it from buses *f* and *g*. Splitting enabled us to write a set of linear equations for the bus.

Maximize

$$r_{2,b}x_{2,b} + r_{3,b}x_{3,b} + r_{b1,b}x_{b1,b} + r_{b4,b}x_{b4,b} \tag{4.32}$$

Equality constraints

$$\begin{aligned} q_{2,b}x_{2,b} = {} & q_{2,2,b}x_{2,b} + q_{3,2,b}x_{3,b} + q_{b1,2,b}x_{b1,b} \\ & + q_{b4,2,b}x_{b4,b} \end{aligned} \tag{4.33}$$

$$\begin{aligned} q_{3,b}x_{3,b} = {} & q_{3,3,b}x_{3,b} + q_{2,3,b}x_{2,b} + q_{b1,3,b}x_{b1,b} \\ & + q_{b4,3,b}x_{b4,b} \end{aligned} \tag{4.34}$$

$$\begin{aligned} q_{b1,b}x_{b1,b} = {} & q_{b1,b1,b}x_{b1,b} + q_{2,b1,b}x_{2,b} + q_{3,b1,b}x_{3,b} \\ & + q_{b4,b1,b}x_{b4,b} \end{aligned} \tag{4.35}$$

$$\begin{aligned} q_{b4,b}x_{b4,b} = {} & q_{b4,b4,b}x_{b4,b} + q_{2,b4,b}x_{2,b} + q_{3,b4,b}x_{3,b} \\ & + q_{b1,b4,b}x_{b1,b} \end{aligned} \tag{4.36}$$

This set of equations is for bus b in Figure 4.12. In this set, bus b is a shared resource between the two processors 2 and 3, as well as buffers *b1* and *b4*. The state of the system is actually given by a bus-processor pair, but since we are writing equations only for bus b, we considered only the processor using bus b as the state of the system. After solving the CTMDP for this system of equations, the state-action pair probabilities were translated into buffer space requirements by using the KSwitching policy for a certain processor-bus pair. Then, the system is simulated with the new buffer lengths, and data losses are recorded, as shown in Figure 4.3.

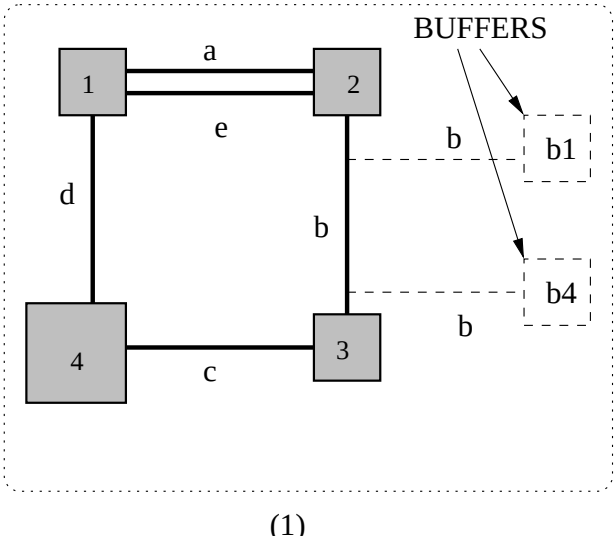

Figure 4.12: **Subsystem 1 for bus architecture in Figure 4.11**

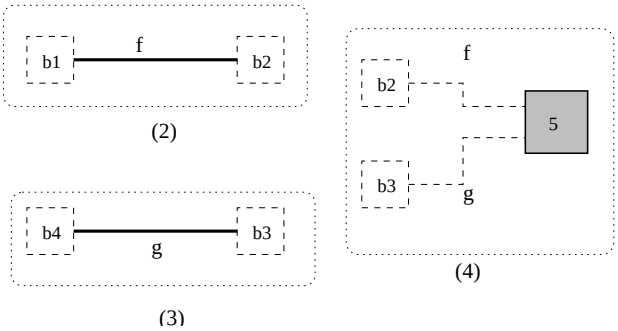

Figure 4.13: **Subsystem 2,3, and 4 bus architecture in Figure 4.11**

4.7 Preliminary Experiments with Queuing Models

We have considered the "Virtex-2 Pro" reconfigurable processor-embedded FPGA . The "Virtex-2 Pro" can have upto 4 "PC 405 Power PCs" communicating with reconfigurable IBM Core Connect bus architecture.The architecture is that of four processors communicating with a bus and each one of them have buffers into which they write requests these requests are then serviced by the bus. The target architecture could be presimulated with different arbitration policies the best of which could be mapped to the reconfigurable " Virtex 2 Pro" processor-embedded FPGA.

The bus has an arbiter which looks at the queue lengths and/or time spent by a request while waiting in a queue to decide which processor should get the service of the bus. The arbiter has the following policies:

- Policy (A) Queue length: Looks at the number of requests in the queues and selects the longest queue to be serviced if there are two or more longest queues of equal length it will select one at random.
- Policy (B) Time spent: Looks at the time spent by a request in the queue and will give the bus to the request that has waited longest. If there are two or more requests with equal longest waiting time it will select one at random.
- Policy (C) Length+Time: Looks at the number of requests in the queues and selects the longest queue. If there are two longest queues of equal length then it will look at the time spent by the request in those two and will allocate bus to the one with longest wait time. If the wait times turn out to be equal too then it selects at random.
- Policy (D) Packetise: Looks at the number of requests in the queues and collects requests untill a packet of requests have been formed. This packet is then served by the bus. The bus arbitration is as in Policy A.

These policies have been implemented on the model with finite population. The parameters studied and enumerated in Table 4.7 are the average queue length for the queues (AVGQLEN[i],i = 1..4 is the number of procesors), the maximum queue length (MAXQLEN[i], i=1...4), the average loss rate for each queue (AVGLOSS[i],i=1...4), the average power consumed in each policy,the total number of requests serviced (COUNT) and the time taken to service them (TOTAL-TIME), which is not in units of physical time but is the time step used in the simulation. A crude throughput in the form of a ratio of count over time has also been shown (CNT/TIME). While calculating power consumption each request has been assigned one unit of power and every time the bus goes to a different processor the switching power has been assigned 0.1 units of power.

4.7.1 Discussion

The queues for the different processors behave differently due to the arrival rates associated with the different processors are different this brings about the need to analyze the behavior of each processor under each policy. A general

Table 4.1: **Comparison of heuristic policies for architecture 1**

Parameter	Policy A	Policy B	Policy C	Policy D
(1)	(2)	(3)	(4)	(5)
AVGQLEN-1	0.1280	1.0924	0.3720	0.1252
AVGQLEN-2	0.0623	0.1512	0.1816	0.0615
AVGQLEN-3	0.0274	0.0440	0.0802	0.0270
AVGQLEN-4	0.1292	0.2137	0.6186	0.1290
MAXQLEN-1	2	7	4	2
MAXQLEN-2	2	5	3	2
MAXQLEN-3	2	2	3	2
MAXQLEN-4	2	1	3	2
AVGLOSS-1	0.0030	0.6361	0.0000	-
AVGLOSS-2	0.0005	0.2423	0.0000	-
AVGLOSS-3	0.0005	0.4369	0.0000	-
AVGLOSS-4	0.0030	0.1509	0.0000	-
POWER	1.2347	2.1442	1.2222	1.2186
COUNT	3310	2107	3332	3333
TOTAL-TIME	467.15	730.96	421.89	492.71
CNT/TIME	7.0900	2.8800	7.9000	6.76

characterization of the policies has been given below the exact behaviour has been displayed through Table 4.7.

- Policy (A) Queue Length: The policy displays a very good max queue length .For example in case of the "Virtex 2-Pro" the worst case scenario is that the queue length is going to 2, so the buffer size could be fixed at 2 and has a good throughput but suffers from loss of data.

- Policy (B) Time spent: The policy performs worse than other policies in almost all aspects so we conclude that looking at wait time in arbitration while trying to use a finite storage, loss system is not good as looking at the number of waiting requests in a queue for arbitration.

- Policy (C) Time+Length: This policy is best in terms of throughput but is not very good in case of max queue lengths.

- Policy (D) Packetise: This policy has the smallest power consumption and moderate throughput. This policy doesn't have any loss associated with it as the bus starts servicing only once there are N requests waiting

in the queues unlike the other policies where requests are continuously serviced and are lost if they arrive and find the system full.

The Packetise policy has the lowest power consumtion and the lowest loss rate as the requests are combined into packets and then sent, but this also results in a large delay in the buffer and the buffer sizes are large too. The TimeSpent policy on the other hand has a very small queue length but a large loss rate and the reason for the small queue length is not the eficiency of the time spent policy but the large loss rate. The Length and the Time + Length policy can behave in a similar fashion and have average performance in all the performance metrics.

Thus, depending on which metric is important to the designer he can make a choice of policy if the policy assigned is static for some of the metrics the policies could be chosen on the fly depending on the state of the system. The size of buffers, the idle time of a shared resource and average loss rate of a system are some of the parameters one could estimate from these simulations. The bottlenecks in a system could be found out by observing which nodes in the stochastic model suffer from loss or congestion and a proposed alternative architecture to remove the bottleneck could be simulated and compared with the one suffering from loss or congestion. This paper has used only deterministic policies for arbitration in future works we shall study randomised policies for arbitration as well differing approaches to obtain such policies.

4.8 Experiments to Compare Heuristic and Stochastic Policies

All the policies discussed were implemented on the queueing models for two architectures. Both architectures have implementations with and without redundant paths between the processors. The experiments have been performed for the bus architectures shown in Figure 4.6 Figure 4.14 and Figure 4.17 The queueing models for the first architecture have been shown in Figure 4.7.The first architecture is a preliminary example which we used to study and explain the technique, and test it before applying it to a more complex real world application .The second architecture is an IBM Network Processor with 3 buses a processor and a number of peripherals and controllers which manage the communication with the outside world as well as in the Network Processor itself . The architectures for the IBM Network Processor have been obtained from [99] and [100]. The queueing models were built by using connectivity information from the architecture and the arrival rates of the requests were

determined from the core graph [99] of the IBM Network Processor . The constraints discussed in the previous section deal with queue lengths and the time spent in the queues. The reward function incorporates the steady state as well as the transition power consumption. We have compared redundant and non redundant bus topologies as well as compared heuristic and stochastic policies. Several sets of results have been displayed in Appendix 1. Concise versions of the results have been displayed as figures and tables through the following sections.

The MDP for the network processor had fifteen states and upto four actions depending on presence of redundancies. The LP solver used was the "linprog" command in matlab. It took 7-8 iterations to solve the equations for the simple architecture and took 22-23 iterations to solve the set of LP equations the difference in number of iterations of the LP solver was negligible as model. We make this statement to show the feasibility of real world applications.

The experimental results obtained after simulating the policies on the queuing model of the architectures consist of the power consumption, the average and maximum queue lengths encountered and the maximum time spent by a request in a queue. Confirming our intuition the heuristic policies consume more power as they are oblivious to the power consumption.The comparison of the policies has been done over different traffic environments. The experiments were performed for exponentially distributed arrival times and exponentially distributed uniformly distributed and deterministic and departure times. Figure 4.15 is the legend for the policies that have been implemented and studied. The stochastic policies consume lesser power yet have small queue lengths. In case the heuristic policies do manage the queue lengths well then the power consumption in them is very large as in Time + Length policy. The Average and Maximum queue lengths are lesser in the cases with redundancy than the ones without. The results closest to intuition were obtained for exponential arrive and depart times as observed in the exponential distribution. Though the results for the policies arent as good as the exponential arrive and depart time case,the cases with Deterministic arrival and departure rates ,exponential arrivals with deterministic or uniform departures also show improvement with introduction of redundancy use of stochastic policies, this is due to the exponential arrival rate causing markovian arrival and departure processes as in Figure 4.16. The behaviour of the heuristic policies is similar in the case with the redundant paths and without redundant paths as observed in Figures 4.16 and4.18 this is becaues these policies can be seen as a class of deterministic policies. They see a certain state in terms of the queue lengths or the delays in the system and react in a deterministic fashion to it.

The stochastic policies did not do well in the deterministic traffic environ-

Table 4.2: **Comparison of policies**

ARRIVE TIME DISTRIBUTION	DEPART TIME DISTRIBUTION	No Redundancies BEST POLICY	Redundancies BEST POLICY
EXP	EXP	3 (HEURISTIC)	4 (STOCHASTIC)
EXP	UNIF	3 (HEURISTIC)	4 (STOCHASTIC)
EXP	DET	3 (HEURISTIC)	4 (STOCHASTIC)

ment especially when there were no redundancies in the architecture as seen in Figure 4.14. In the case without redundancies the stochastic policies fail to perform well because in the case without redundancies the all 3 stochastic policies will reduce to the same policy as there is only one action available at every state. In the graphs for the non redundant case we see that the randomised stochastic policy as well as the deterministic stochastic policy behave in a similar fashion whereas in the case with redundancies the randomised policy does outperfom the the deterministic stochastic policy as it uses the alternative actions available to it at every state. Though the K-Switching has large power consumption in the case without redundancies it does manage to give very good control over the queue lengths.In the case with redundancies the K-Switching appears to be optimal among the policies discussed and gives low power consumption with small queue lengths. The deterministic policy consumes about the same power as the randomised policy but since it chooses the same action in a state it tends to have a larger maximum queue length. The K-Switching cycles through every action in a state and if the constraints are not very conservative, then it may ignore them and greedily give maximum time to the least power consuming state action pairs. In the case without redundancies the power consumption in the Randomised stochastic policy as well as the Deterministic policy is high. In Table 4.8 we list the best policies among the policies we studied for the different traffic scenarios for architectures with and with out redundancies. Please refer legend for the policy names.

4.8.1 Experiments for Bus Arbitration Policies

The first set of experiments compared the stochastic bus arbitration policies produced using the proposed methodology with heuristic policies. Stochastic policies were produced to minimize power consumption under maximum queue length constraints. The reward function used in the stochastic models incorporated the steady-state power consumption as well as the transition power consumption. Experimental results were observed for the resulting power consumption, the average and maximum queue lengths encountered,

and the maximum loss experienced in a certain policy. Figure 4.15 is the legend for the arbitration policies that have been implemented and studied.

Comparisons have been done over different traffic environments, including exponentially distributed arrival times, and exponentially distributed, uniformly distributed, and deterministic departure times. Such queues pertain to the M/G/1/K category [48]. Similar distributions have been considered by several recent work. [8] uses M/M/1/K queues for modeling interconnect channels in integrated circuits. These queues are a subset of M/G/1/K queues, and correspond to exponential arrival and departure times. [76] uses uniform, exponential, and normal traffic for simulating disk access interarrival times. Combinations of more than one exponential distributions are used in [75][76] to capture non-exponential interarrival times.

All bus arbitration policies were implemented on the queuing models for bus architectures without redundant and with redundant paths between the processors. The used bus architectures are shown in Figure 4.14, and Figure 4.17. The two architectures are for an IBM Network Processor with three buses, a processor, and a number of peripherals and controllers. The two bus architectures have been obtained from [99] and [100]. The queuing models were built and simulated by using connectivity information from the architecture. The arrival rates of the requests were determined from the core graph [99] of the IBM Network Processor.

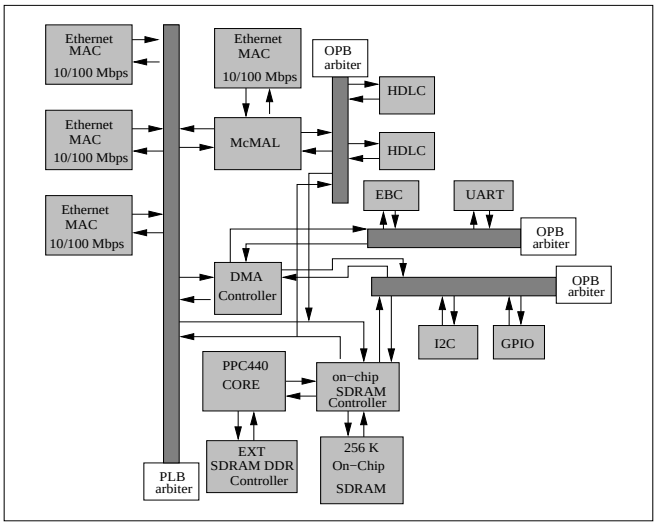

Figure 4.14: **Network Processor architecture without redundant paths**

Figure 4.16 presents the experimental results obtained for heuristic and stochastic policies applied to the architecture without redundant paths, and a

LEGEND FOR X−AXIS (POLICIES)

(1)−TIME SPENT POLICY (2)− RANDOMIZED STOCHASTIC POLICY
(3)−TIME + LENGTH POLICY (4)− K−SWITCHING STRATEGY
(5)− ROUND ROBIN POLICY (6)−DETERMINISTIC POLICY

Figure 4.15: Legend

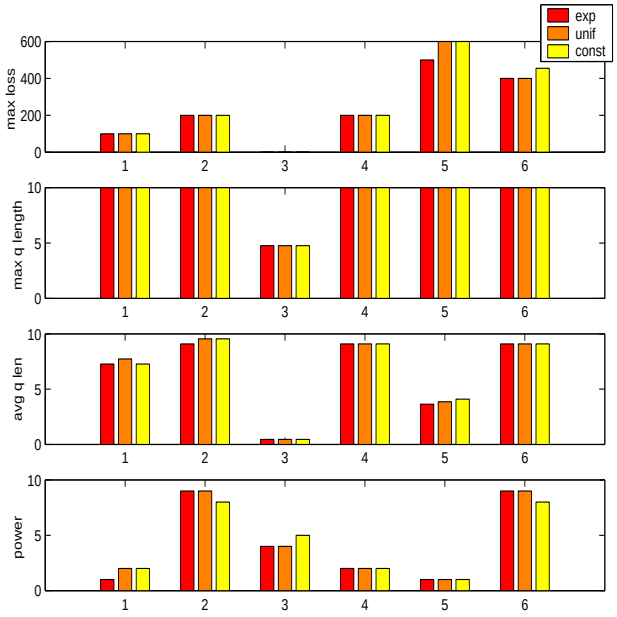

Figure 4.16: **Results for Network Processor without redundant paths**

buffer length constraint of 160 entries in the buffer. This is a typical size for a fast set of data registers. Figure 4.18 shows the results for the architecture with redundant paths and same buffer size constraints. For non-redundant bus architectures, round robin policy offered the lowest power consumption, but its data loss was very high. Time + length policy had higher power consumption, but its data loss was minimal, and maximum queue length was small. Among heuristic policies, KSwitching policy resulted in the lowest power consumption and data loss. The maximum queue lengths were similar in all stochastic cases. For redundant bus architectures, KSwitching policy offered the lowest power consumption and data loss. Also, the maximum queue length was smaller than in the other cases. Among heuristic policies, time + length policy offered fairly good results, including a very small data loss. All policies, but KSwitching, had a large maximum queue length. Similar results were obtained for buffer length constraints of 480 and 640 elements.

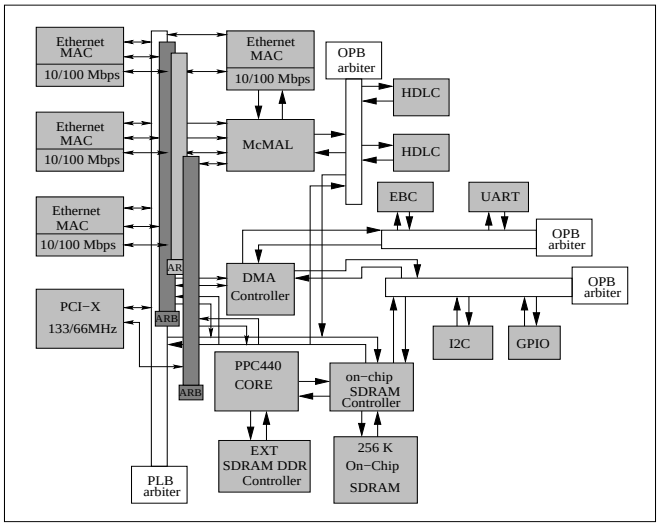

Figure 4.17: **Network Processor architecture with redundant paths**

The results for the different traffic environments have been discussed below. The behavior of the policies was similar in cases of exponential arrivals with exponential or uniform departures. Hence, the discussion has been split over two sub-headings, (1) one for the case with exponential arrivals and exponential or uniform departures, and (2) the other for exponential arrivals with deterministic departures.

- *Exponential arrivals and deterministic departures:* Stochastic policies did not do well in the deterministic traffic environment, especially when there were no redundancies in the architecture, as seen in Figure 4.16. This is because all three stochastic policies reduce to the same policy, as there is only one action available at every state.

- *Exponential arrivals and exponential or uniform departures:* Figure 4.16 shows that for the non-redundant case, the randomized stochastic policy and the deterministic stochastic policy behave in a similar fashion. Though the KSwitching policy has large power consumption in the case without redundancies, it gives very good control over the queue lengths.

  In the case with redundancies, the KSwitching policy performs best among all policies. It gives the lowest power consumption with a small queue lengths. The KSwitching policy cycles through every action in a state, and if the constraints are not very conservative, then it may ignore them, and greedily give maximum time to the least power consuming state-action pairs. In the case without redundancies, the power

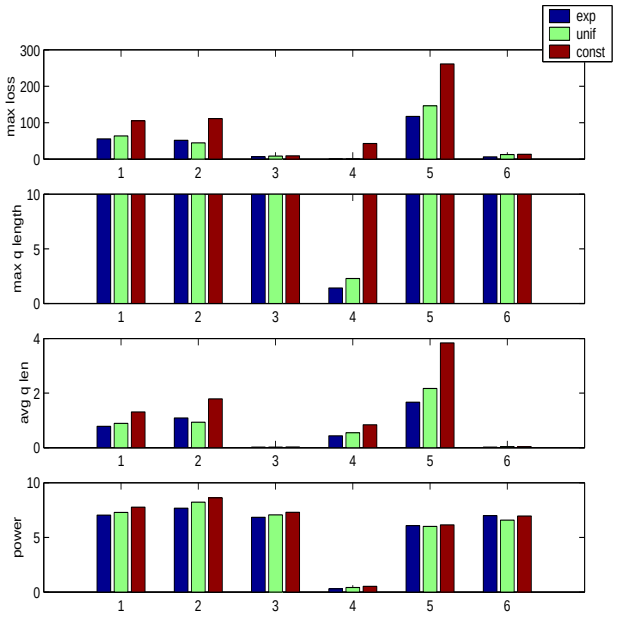

Figure 4.18: **Results for Network Processor with redundant paths**

Table 4.3: **Loss at processors under varying total buffer size**

PROCESSOR	Buf 160		Buf 320		Buf 640	
	pre	post	pre	post	pre	post
1	70	83	41	40	48	0
4	80	100	78	55	74	0
15	107	90	99	12	88	0
16	96	82	84	0	93	0

consumption in the randomized stochastic policy and the deterministic policy is high.

4.8.2 Experiments for Buffer Sizing

These experiments used a network processor [99] as a test architecture for buffer insertion and buffer space distribution. It provides opportunity to explore the different scenarios of buses talking to other buses, as well as buses that talk only to processors, and do not need explicit insertion of buffers. The architecture has been shown in Figure 4.19. Bridges are connections between buses, and need buffers to be inserted apart from all buses, which have buffers to handle the traffic coming in from the processors.

Table 4.4: **Total loss under varying total buffer size**

TOTAL BUFFER SPACE	TOTAL LOSS	
	pre	post
320	580	207
368	576	161
400	557	47
432	566	2
480	535	0
512	517	0

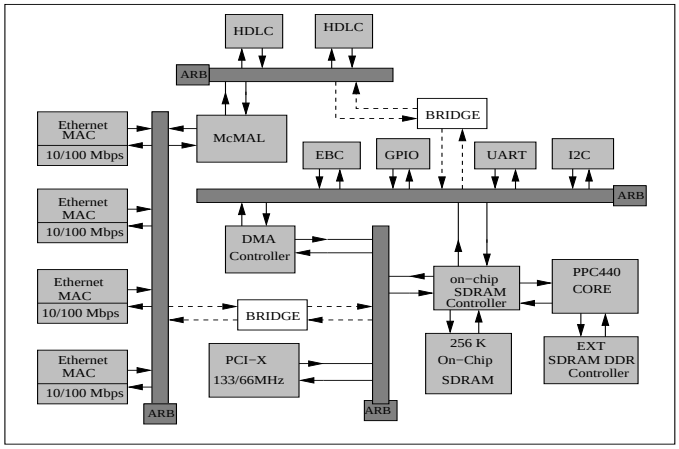

Figure 4.19: **Network Processor bus architecture with buffers and bridges inserted**

Figure 4.19 shows the Network Processor architecture with buffers inserted as arrows, and the buffers between buses and bridges as dotted arrows. In Figure 4.20, we have plotted the loss rates at the processors before and after the buffer sizing, as the first and second bars of Figure 4.20. We found that though the loss rates decreased drastically for some processors (such as for example processor 16), they increased slightly for some processors, e.g., for example processor 1. However, the overall loss decreased. The third bar in Figure 4.20 are the loss rates for timeout policy. In this policy, the processors request is not served, if the data in the buffer times out, i.e. reaches a threshold time. The threshold time chosen was the average time spent by a request in a buffer.

We repeated these experiments for ten iterations, and found that though the loss may increase for some processors, the overall loss of the system decreases by about 20% as compared to the constant buffer sizing policy, and

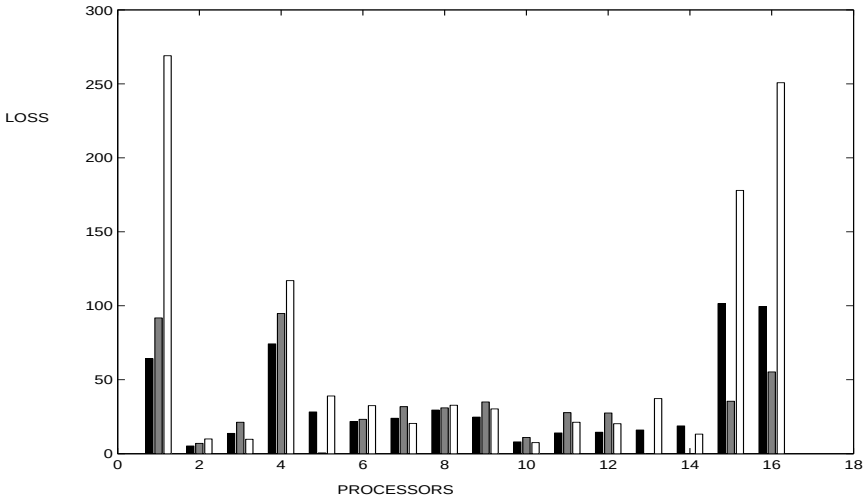

Figure 4.20: **Loss rates at processors before and after sizing the buffer space**

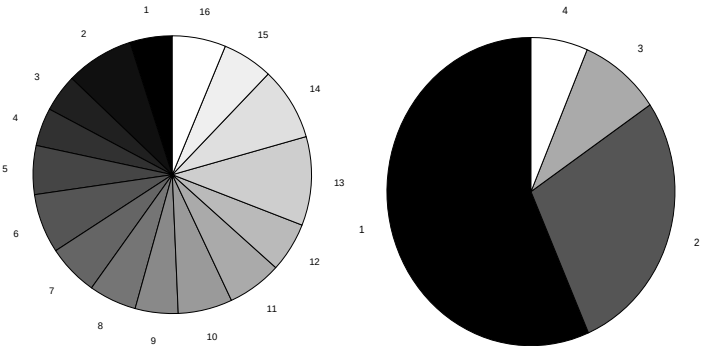

Figure 4.21: **(1) Distributions among processors, and (2) buses**

50% for the timeout policy. We feel the difference before and after resizing could be improved with better profiling and weighing of the loss at processors, i.e. allowing some losses to be more important than the others.

Table 4.8.1 presents the variation in the loss rates before and after sizing the buffers. We have presented the results only for a few processors, which show significant variation. However, a similar trend was observed for the rest of the processors. We observed that some processors loss rates may increase when the buffer space is very limited, as in the 160 units case. In this case, the redistribution doesn't provide much improvement. We increased the total buffer space from 160 units to 320 units, and then to 640 units. The loss rates after resizing decreased with the increase in buffer space, and was zero for

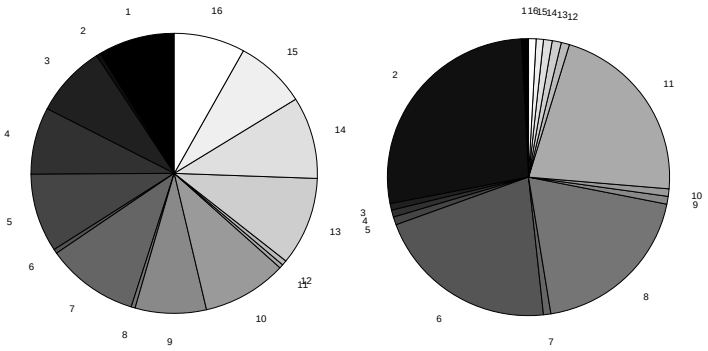

Figure 4.22: **(1) Distributions at Bus 1, and (2) Bus 2**

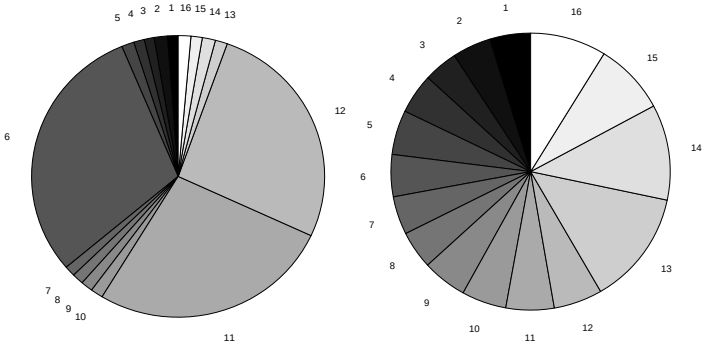

Figure 4.23: **(1) Distributions at Bus 3, and (2) Bus 4**

buffer space of 640 units.

Table 4.8.2 compares the total loss before and after sizing, while the total buffer space available was increased. The total loss decreased with and without sizing, as we increase the total buffer space. The decrease in the case with sizing according to the KSwitching policy is much sharper, as shown in the third column of Table 4.8.2. For buffer space values beyond 480 units, the loss after resizing remained zero, while the total loss before sizing decreased gradually.

The distribution of the buffer space being our main concern, we observed the distribution of the buffer space of 160 units among the 16 processors, as shown in Figure 4.21 (left), as well the distribution of the space among the four buses in Figure 4.21 (right). We also looked at the buffer space distribution at each of the buses. The distribution of buffer space for buses 1 and 2 is in Figure 4.22, and for buses 3 and 4 in Figure 4.23. The distribution of space among the buses as well as the distribution of space at the individual buses among processors does not match the ratio of traffic in the simulation model of the bus architecture. This is due to the connectivity provided by the bus

architectures modifying the traffic.

4.9 Summary

Using the the stochastic modeling based environment we generated arbitration policies and compared the policies over several parameters. The stochastic policies could outperform the heuristic policies, and the Randomised policy and K-Switching policy in particular tend to give better results than the rest of the policies in terms of power and delay and power vs quelength trade off. The arbitration policies could give fair and starvation free arbitration due to the extra bounds on the LP. The MDP based methods can be used to solve difficult arbitration problems in practical scenarios.

Chapter 5

Continuous Time Adaptation in Processing Subsystem

Sensor networks are emerging as a revolutionizing technology for many important applications in national security, health care, environmental monitoring, infrastructure security, food safety, manufacturing automation and many more [60, 77]. In fact, the vision is that sensor networks will offer ubiquitous interfacing of the physical environment to centralized databases and computing facilities [109]. Efficient interfacing has to be provided over long periods of time and for a variety of environmental conditions, like changing temperature and weather conditions, variable amounts of energy, presence of moving objects and so on. In this context, a key problem that ought to be tackled is that of devising embedded software and hardware architectures that can effectively operate in continuously changing, hard-to-predict conditions. In addition, architectures should be cheap and energy sparing - considering that their batteries are hard to be replaced or replenished.

For moving vehicle tracking, one of the main applications of sensor networks, the vehicle's velocity, trajectory and position defines the required sampling rate, hence the throughput constraint for image processing [109]. Varying throughput constraints necesitate continuous adaptation of a sensor node's architecture, including selection of the supply voltage and clock frequency, allocation of hardware resources to software tasks and reconfiguration of functional blocks. Architecture adaptation for vehicle tracking is challenging because static, off-line prediction of a vehicle's movement is quite inaccurate in real-life. Even if vehicle movement is predictable with a certain accuracy, the resulting off-line model is highly non-linear and discontinuous in many points. Therefore, it is quite inefficient to address this architecture adaptation problem by typical embedded design methods [21, 46, 76], which consider static, quasi static or stationary scenarios that are described through fixed, off-line models.

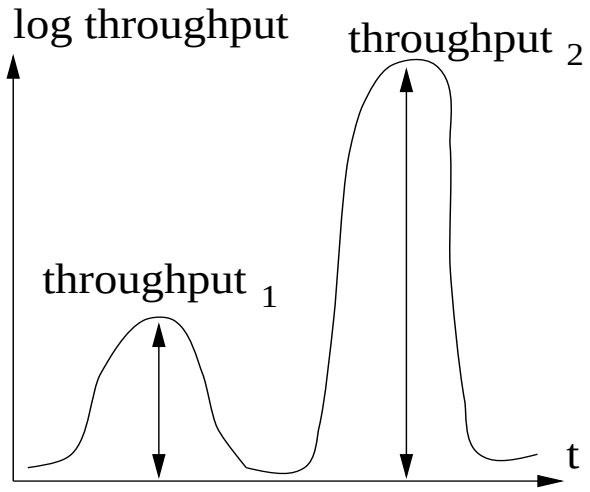

Figure 5.1: **Throughput variation for tracking**

As explained in the paper, vehicle tracking requires expression of uncertain performance requirements, on-line model identification as well as a scheme for continuous architecture adaptation to varying performance needs.

5.1 Motivation

Sensor nodes operate in environments with continuously changing attributes. Figure 5.1 presents a fragment of the throughput constraint variation plot that we experimentally found for a sensor node that tracks one moving object. More results are presented in Section 5. The throughput requirement of a node is highly dependent on the trajectory, speed and proximity of the moving object. Please note that the speed requirements at time moments t_1 and t_2 are about 10 and 100 times higher than in the steady state - when the tracking activity is low. In fact, requirement variations are even higher, if several fast moving vehicles are simultaneously monitored. Other changing attributes include the amount of energy stored in a sensor node's battery, the topology of a sensor network, the temperature of the environment, the air humidity, and so on.

Designing the software and hardware architecture of sensor nodes is essentially different from that of traditional embedded applications due to the need to accommodate performance requirements that rapidly change in a wide range. For our example, the architecture should be capable of adjusting in less than 20 units of time to speed requirements that are about 50 times larger. Also, most of the time these systems operate in environments that are hard to predict. In spite of some rough predictions that can be made, in real settings,

the actual moving of vehicles is hard to anticipate. There are simply to many factors that influence it, like time of the day, sensor node positioning, road and weather conditions, and much more. Therefore, it is hard to estimate the best-average-worst case situation in which the architecture will *actually* operate and optimize it for these cases. Finally, the wireless network topology that these nodes form is ad-hoc and highly dynamic, and thus variable computing and communication loads will be allocated to the individual nodes. In conclusion, sensor node architectures do not fit the typical embedded system design paradigm, in which hardware and software are co-optimized for a static or predictable set of performance constraints [21, 46, 76]. Instead, optimization should be conducted on-line - during execution, in response to punctual performance requirements in an environment hard-to-predict.

An immediate consequence of these differences is that sensor node architectures should not be optimized off-line using worst-case design approaches. Reason is that - even for simple cases in which the worst case is predictable, it would lead to over-design with unfeasibly high costs and energy consumptions. For example, optimizing the system for the maximum $throughput_2$ in Figure 5.1 would mean that the architecture is not really customized to the application for most of the time. Even worse, over-design is further amplified, if the architecture runs multi-tasking applications for which worse case situations rarely occur simultaneously. For example, a high throughput requirement for the face detection algorithm [109] does not imply that data compression and communication should be fast also. If the moving vehicles are not of interest then there is few data to be sent to the central server. Besides a much larger cost, the hardware would also consume more energy due to larger controller and decoder circuits - even if the unused functional units are shut down. For sensor nodes, the area constraint is essential as various RF, analog, digital and memory circuits need to be squeezed within a tiny silicon area. Having more transistors intuitively represents more energy waste due to leakage currents. Hence, on-line optimization should continuously minimize the amount of unused resources in addition to minimizing energy consumption and meeting throughput constraints. Minimizing power consumption through minimizing the amount of unused resources, thus the architecture over-design, is similar to [35].

The thesis proposes a procedure for modeling continuously changing throughput requirements and then describes a technique for selecting the optimal set of design points (DP) that accommodate variable throughput constraints at the expense of minimum energy consumption. The proposed methods are discussed in the context of image processing for face (moving object) detection in sensor networks [109]. In our experiments, each DP represented a certain

clock frequency and supply voltage of the sensor node processor, even though the methods support a more general formulation of architecture adaptation, including hardware reconfiguration or task repartitioning [56]. We explained that continuously varying throughput constraints can be *collectively* expressed using *performance bands* with linear, quadratic and exponential variation depending on the vehicle movement characteristics, like velocity, trajectory and position. Then, any actual throughput variation is contained in a band. Assuming that each throughput variation within a band has the same probability of occurring, we presented an analytical procedure for selecting the optimal set of DP for different kinds of performance bands. Energy consumption is modeled using a metric called over-design factor similar to Henkel [35]. Experiments study the effectiveness of performance modeling and DP selection procedures.

On-line optimization methods, including on-line software optimization and run-time hardware reconfiguration [33, 56, 113], fall into two categories: (1) methods that select on-line from a set of optimal alternatives computed off-line [50, 56, 85, 86, 116] and (2) methods that conduct on-line optimization [4, 73] , e.g., Dynamo - a software optimization method proposed by Bala *et al* [4]. The proposed DP selection technique pertains to the first category, and is original in that it focuses on design for continuously changing, hard-to-predict performance constraints. Existing work on dynamic partitioning for reconfigurable hardware suggests dynamic selection from a set of binary or source code that is sent to the hardware modules [85, 86]. However, this work does not explain how the set of alternative binary code should be decided. Other approaches are for reducing energy consumption by dynamic resource allocation [64, 46]. They predetermine the hardware configurations for certain static design requirements, and then search among them by using heuristics to improve energy consumption. Finally, voltage scheduling reduces power consumption by dynamically selecting the processor supply voltage from a predefined set [3, 116]. The main contributions of this paper are in proposing a procedure for modeling continuously varying throughput requirements as well as an analytical method for computing the optimal set of design points for dynamic performance constraints.

As in the case of the communication subsystem the processing subsystem also needs continuous adaptation in presence of varying QoS requirements and available processing resources. The state of the processing subsystem could be defined as the amount of active resources that are being used for computation at a given instant of time. An example of a system that uses differing numbers of architectural resources has been discussed in [107] and [108]. In this effort the available hardware resources have been made available in terms

of the number of processing elements of various types as well the HW/SW partitioning of that set of processing elements. The application in consideration was a face recognition sensor. The hardware resources have increasing levels of processing element numbers in Table 5.1 and these configurations when used can provide satisfaction to certain timing constraints. We have used a part of the table in [108]. In case a timing constraint is not satisfied then a configuration with higher number of processing elements could be used. As the processing elements have been shared across the tasks allowing high utilisation of the elements in parallel to complete a task faster. Thus the configurations with higher resources would be able to satisfy much tighter timing constraints. The various timing constraints that could be satisfied by certain processing subsystem architecure have been listed in Table 5.1.

Table 5.1: **Possible resource configurations [108]**

Resource sets	Adders	Multipliers	NAND Gates
Set 1	128	128	512
Set 2	512	128	512
Set 3	128	512	512
Set 4	512	512	512
Set 5	1024	1024	512

The states and transitions of the models for the architectural reconfiguration have been shown in 5.2 . The Markov Chain shows that each state can transition to every other state. The reason all such transitions are available to the system is that though most of the time the transitions will be like a birth death process only to the neighbhouring state sometimes changes may be sudden and drastic requiring transitions to states which are very different from the current state. An example of when such drastic changes could be needed is if the QoS is user controlled like in present day video streaming applications where there is a choice of video quality (low,medium,high).

5.2 Problem Description

Figure 5.3 shows a simplified presentation of the sensor node architecture. The node tracks the environment using a camera. The predictor block conducts on-line estimation of the performance requirements for the architecture (in our case system throughput). Based on dynamic prediction, the controller

Table 5.2: **Constraint Satisfaction and Reconfiguration conditions [108]**

Tasks	Set 1	Set 2	Set 3	Set 4	Set 5
Cr	<588	<425	<365	< 365	< 192
Crth	<440	151-513	61-151	32-61	<32
Cb	<320	320-365	365-425	192-365	<192
Cbth	<440	440-513	61-151	32-61	<32
Gmean	<155	155-180	41-52	52-78	<52
Gfmean	<17	<20	<17	<21	<22
Cov	<301,500	<218,000	<255,590	<24,518	65,536
Sum	<245,000	<218,000	<255,590	255,590-311,296	311,296-327,680
Th	<147	<170	<50	<61	<32
Pick	<282,600	282,600-326,860	282,600-326,860	326,860-373,555	373,555-393,216

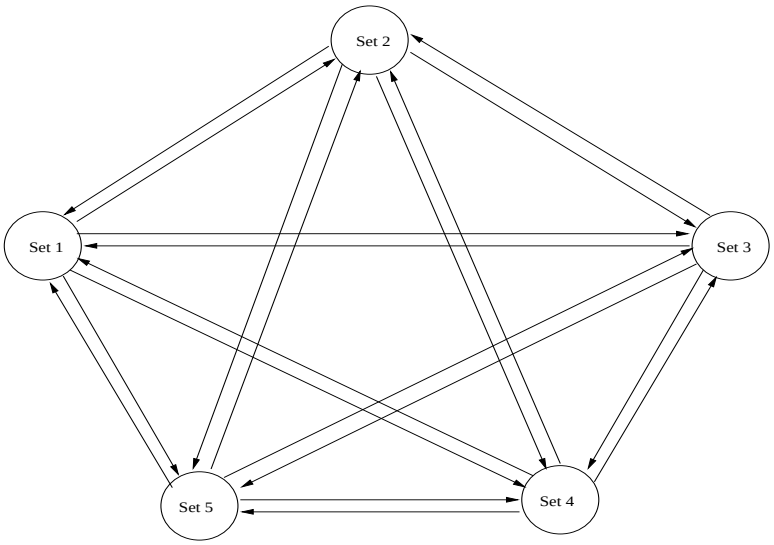

Figure 5.2: **State transition diagram for the Processing Subsystem adaptation**

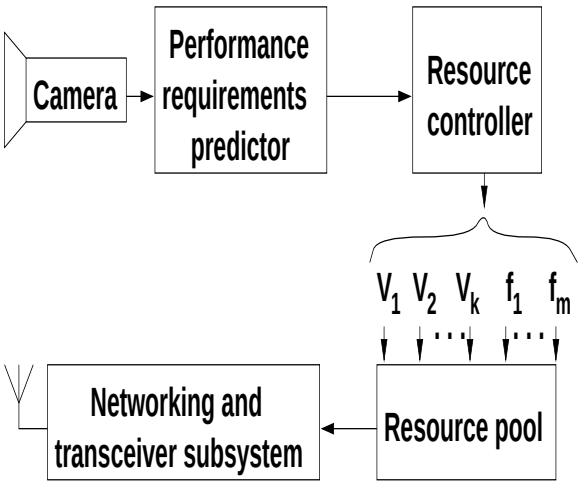

Figure 5.3: **Sensor node architecture**

circuit decides the voltages and frequencies of the hardware resources, including general-purpose processors, executing each task. Processing results are then communicated to other sensor nodes or to the base stations using a networking module followed by a transceiver block.

In this paper, architecture adaptation to dynamic throughput requirements is achieved through modifying the supply voltage and clock frequency of the processors. Voltages v_i and frequencies f_j pertain to a set of discrete values $V = \{V_1, V_2, ..., V_k\}$ and $F = \{f_1, f_2, ..., f_m\}$ respectively fixed for a certain processor [39]. For ASIC, in general, the cardinality of set V is decided by the specific fabrication process as well as the amount of silicon area that is assigned for voltage regulators, level shifters etc. For SoC with multiple voltage islands [38], set V includes all distinct voltage values of the islands. Similarly, the cardinality of set F depends on the complexity of the clocking circuitry, like PLLs, frequency dividers, and so on. In reality, the cardinality of the discrete sets V and F is not too large given the constraints imposed by the electronic circuits used to realize the two sets.

The product $DPS = V \times F$ defines the discrete space of design points (DP) that can be obtained for a certain sensor node architecture. Intuitively, the pair (v_{min}, f_{min}) defines the architecture with least processing speed capability, and the pair (v_{max}, f_{max}) corresponds to the DP with highest computing speed. In fact, the set DPS can be pruned by eliminating the Pareto dominated points, however, this does not change the essence of the presented architecture adaptation methodology. In addition, one could consider a more general definition of the set DPS, including dynamically reconfigurable architectures, reconfigurable processors, on-line software optimization, and so on.

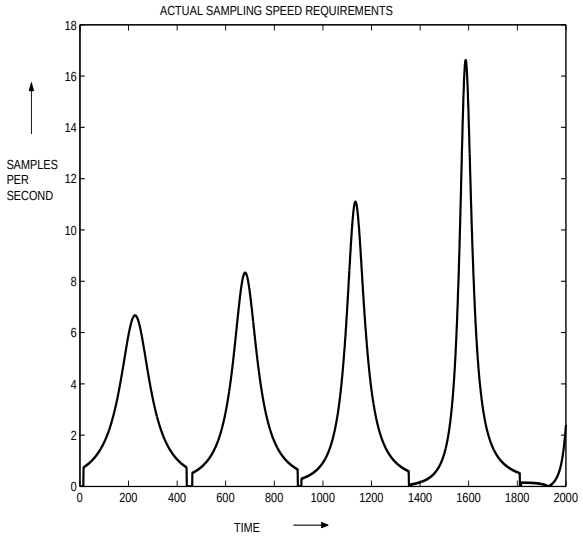

Figure 5.4: **Throughput constraint variations**

The proposed methodology is applicable to the more general definition of set DPS too.

Problem Definition. To simplify the reasoning, we considered that only Pareto optimal design points (DP) are present in the set DPS describing an adaptable architecture. The architecture iteratively executes the same functionality under throughput requirements that change dynamically. The architecture adaptation scheme assumes that the current DP ought to be used at least for the next iteration, and can be replaced only at the end of an iteration. Then, for a given maximum limit LIM on how many of DP can be selected from the set DPS (e.g., due to extra electronics required or the memory space needed to store reconfiguration data or binary code alternatives), the problem is to determine the subset $DPS' \subset DPS$ with cardinality $Card(DPS') \leq LIM$, such that (a) the varying throughput requirements are always satisfied and the (b) the total over-design factor of the architecture over the entire execution period is minimized. The over-design factor (defined in Section 4) expresses power consumption reduction similar to [35].

5.3 Experiments

Experiments studied the (A) modeling power of performance band modeling and (B) over-design factor reduction through design point selection.

A. Throughput constraint variation modeling. We first studied the dependency of throughput variation modeling on the time window length used by the performance requirements predictor block in Figure 5.3. The predictor block performs model identification at the end of each time window to identify the kind of performance bands that should be used. Such an experiment is important not only for understanding the modeling power of performance bands but also for identifying the processing and memory needs for the predictor block.

The throughput constraint curves measured in experiments have been approximated by three different functions over a selection of different time window lengths. The measured throughput requirement (equal to the image sampling speed) has been shown in Figure 5.4. As explained in Section 4, the three functions used are (1) linear, (2) quadratic and (3) exponential. The window length was kept constant over each approximation run, and a curve of the above mentioned three types was selected to approximate the throughput requirement curve over the time window length. The selection for the function was done by minimizing the mean square error between the approximation and the actual throughput constraint curve.

The approximations show some dependency on the time window length. The different window lengths caused a selection of different approximation functions over the same region, as the mean square error seen by the window over a certain region will be different as the window size changes - hence resulting in selection of different approximation functions. Smaller window sizes led to usage of linear segments (thus linear performance bands) in a predominant manner, as shown in Figure 5.5. In the larger window lengths, the fast sampling speed variation suited the exponential and the quadratic performance bands. The exponential functions were used to map in regions where the rate of variation in the sampling speed was very steep as seen in Figure 5.6. The quadratic performance bands were useful to approximate regions where the curvature was large even though the rate of variation in the sampling speed curve was not much. For very large windows the approximations did not look like the actual sampling requirements, as seen in Figure 5.7.

The values of the linear performance band coefficients (a_{min}, a_{max}, b_{min}, b_{max}, c_{min} and c_{max}), which define the performance variation curves have been listed in Table 5.3. The variance of the coefficients was found to increase in general with the number of windows used or with decreasing of the window lengths. In some cases, the approximations with quadratic or exponential functions were not very feasible and the solvers provided solutions, which were close to step functions as the intercept coefficients in the approximation curve were predominant and the coefficients of the higher powers of t or the

Table 5.3: **Coefficients for the linear (lin), quadratic (quad) and exponential (exp) performance bands**

Type	Win	a_{max}	a_{min}	b_{max}	b_{min}	c_{max}	c_{min}
lin	40	0.2360	0.0006	13.9799	0	-	-
exp	40	0.1761	0.0004	14.1561	0.0031	-	-
quad	40	0.0172	0	0.5620	0.0002	13.9799	0
lin	50	0.2081	0.0010	13.9799	0	-	-
exp	50	0.0706	0.0003	14.0505	0.0032	-	-
quad	50	0.0131	0.0000	0.5003	0.0002	13.9799	0
lin	80	0.1498	0.0011	13.9799	0	-	-
exp	80	0.0040	0.0000	13.9840	0.0001	-	-
quad	80	0.0059	0.0000	0.3222	0.0003	13.9799	0
lin	100	0.1265	0.0028	13.9799	0	-	-
exp	100	0.0006	0.0000	13.9805	0.0000	-	-
quad	100	0.0042	0.0000	0.2911	0.0009	13.9799	0
lin	200	0.0691	0.0040	13.9799	0	-	-
exp	200	0.0000	0.0000	13.9799	0.0000	-	-
quad	200	0.0013	0.0000	0.1829	0.0051	13.9799	0

exponential of t were small.

B. Design point selection. The second set of experiments observed the over-design factor (ODF) reduction through DP selection. The obtained performance bands were used to approximate the throughput requirements and calculate the optimal set of DP, as shown in Section 4. For the design points there were ODF, which had to be calculated in order to show the over-design in the system over a particular time window. In the first row (FleDP) of Table 5.4, we showed the characteristics of the ODFs calculated for the optimal DP sets selected by our methodology to find the DP with the minimum ODF. In current practical scenarios, the possible DPs are fixed [39], and are not obtained based on an application, as in our case. The second row (FixDP) in Table 5.4 gives statistics for such fixed design points.

Figures 5.8 and 5.9 show the possible design points for time window lengths of 40 and 80 seconds as calculated by our methodology. Figures 5.10 and 5.11 present the selected design points for the case where the total set of design points is fixed. For the case with fixed design points, we looked at a fixed number of design points in each window, where as for the optimal design point extraction method we stored a variable number of design points depending on how fast the sampling speed was varying in that particular window. Table 5.4 indicates the difference between the ODFs for a window length of 80 as shown

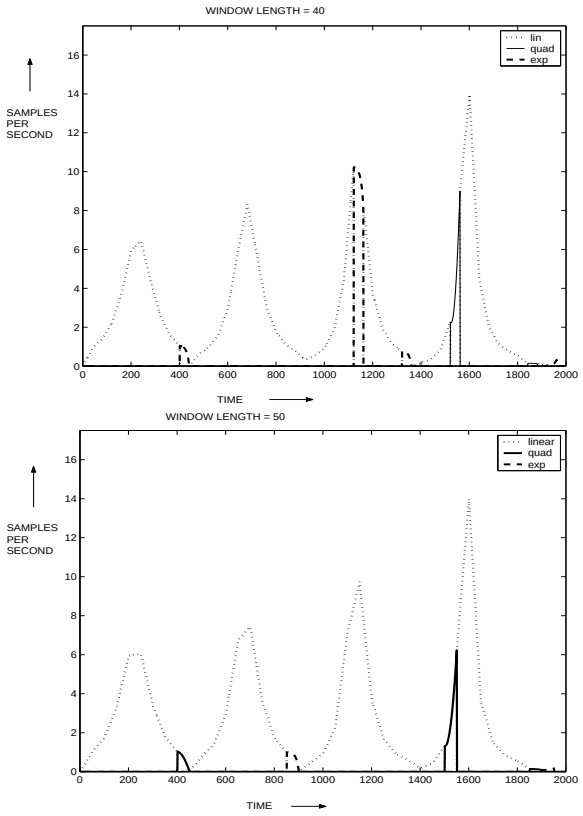

Figure 5.5: **Throughput constraint variation modeling for small time windows**

in figures 5.11 and 5.9.

The design points are defined in terms of the operating frequency of the processor. We have used a face detection algorithm [109] as our application executed on two IBM PowerPC 750 FX processors [39]. The reason we are using two processors is that each one of the processors could be used for a dedicated task, and thus we would have more control over the power saving ability of each processor. The face detection algorithm was split into two main tasks (1) Skin Feature Extraction and (2) Face Template Matching [109], which are carried out in a pipelined fashion. The IBM PowerPC has the ability to

Table 5.4: **ODF for time window length =80**

	DP_1	DP_2	DP_3	DP_4	DP_5	DP_6	DP_7
FleDP	0.2153	0.0069	0.0040	0.0291	0.0315	0.0110	0.0114
FixDP	0.5818	0.0229	0.0317	0.5964	2.1450	0.6121	0.5818

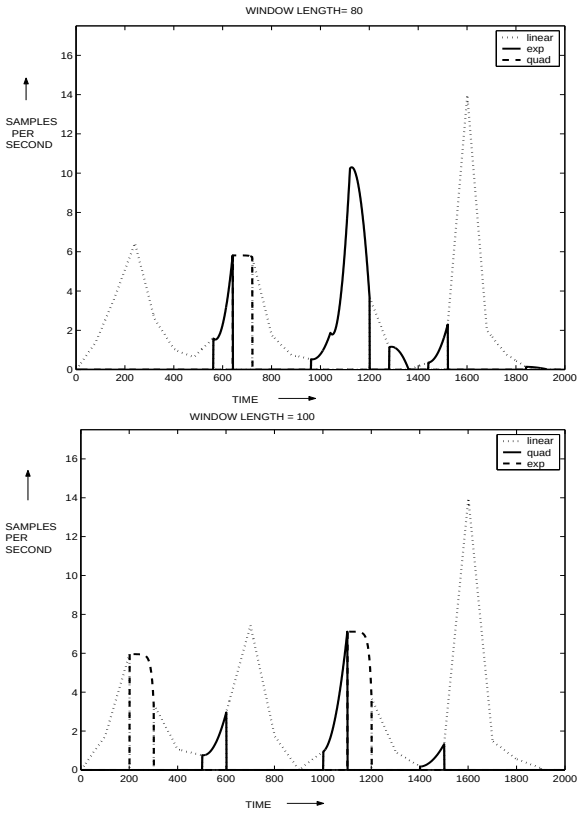

Figure 5.6: **Throughput constraint variation modeling for large time windows**

work at variable voltage and frequency levels [39]. The DPs in our experiments for the fixed design points case vary as 8 discrete levels ranging from f_{max} to $\frac{f_{max}}{16}$. f_{max} is the maximum operating speed, which is needed to handle the worst case timing constraints. The worst case timing constraints occurs at the maximum sampling speed required. As seen from Figure 5.4, this is close to 16 samples per second, which means that 16 iterations of the face detection algorithm would have to be run in one second.

Using the instruction set of the PowerPC and the software specification of the application, we found the total number of clock cycles, which would be required for each iteration of the face detection algorithm. We found f_{max} to be 65 MHz. At situations other than the worst case the PowerPCs shall run at a fraction of this f_{max}, thus benefiting from the relative power saving at the lower frequencies [39]. The two separate tasks have different computation requirements, and the two processors will work at different fractions of their maximum frequency.

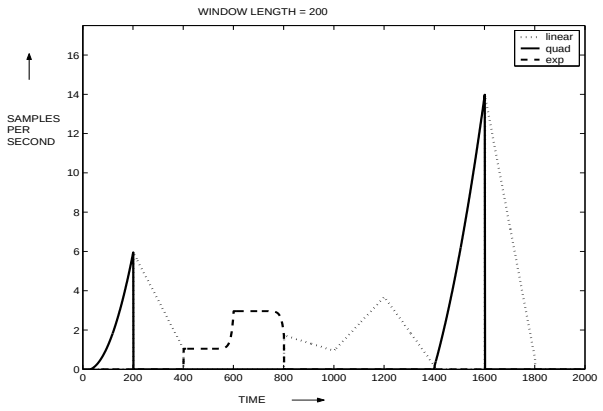

Figure 5.7: **Throughput constraint modeling for very large time windows**

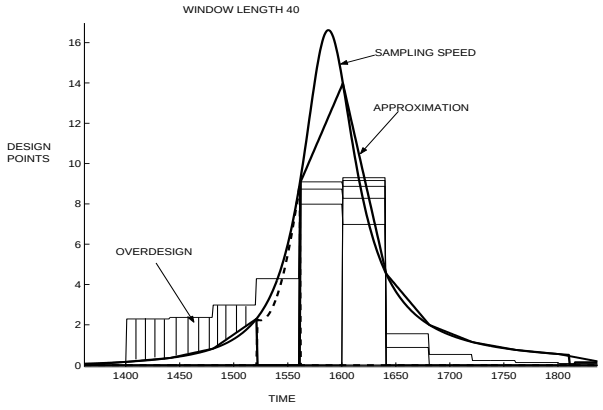

Figure 5.8: **DP selection for small time windows and flexible DP set**

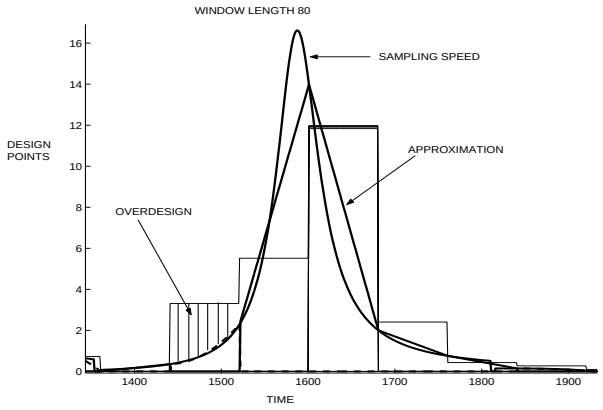

Figure 5.9: **DP selection for large time windows and flexible DP set**

Table 5.5: **ODF for varying Window Lengths**

Window length	fixed DP set			flexible DP set		
	avg	max	min	avg	max	min
40	0.1532	1.0684	0.0100	0.0351	0.0568	0.0013
50	0.1810	1.3861	0.0273	0.0442	0.0729	0.0058
80	0.3380	2.4614	0.0347	0.0848	0.1299	0.0023
100	0.4526	3.1869	0.0589	0.1289	0.2058	0.0116
200	1.1587	6.7278	0.0729	0.3484	0.5297	0.1766

In Table 5.5 we listed the characteristics of ODF values for the design points closest to the curves. We saw that ODFs were larger for the fixed set of design points case as compared to the flexible set of design point case. In the flexible design points case the design points were brought closer to optimal, i.e. having a smaller over design factor. ODFs increase with the window lengths as the approximations get less accurate as the window length increases. The average values of the ODFs were used to approximate the relative power savings between the fixed set of design points and the flexible design points cases. These ODFs correspond to the excess resources being utilized. In our case, the over-design corresponds to the system operating at a frequency greater than that required by the current processing load. The lower frequencies allow usage of lower voltage levels thus allowing further power saving [39] The frequencies range from f_{max} to $\frac{f_{max}}{16}$ and the supply voltage levels vary from V_{dd} to 75% of V_{dd}. At frequency f_{max} only the use of supply voltage V_{dd} is permitted, but at lower frequencies, like $\frac{f_{max}}{2}$ and $\frac{f_{max}}{4}$, one can use 87.5 % of supply voltage V_{dd} and can go still lower to 75% of V_{dd} for frequencies like $\frac{f_{max}}{8}$ and $\frac{f_{max}}{16}$. By doing this we can achieve more than linear saving in power while we reduce the frequency.

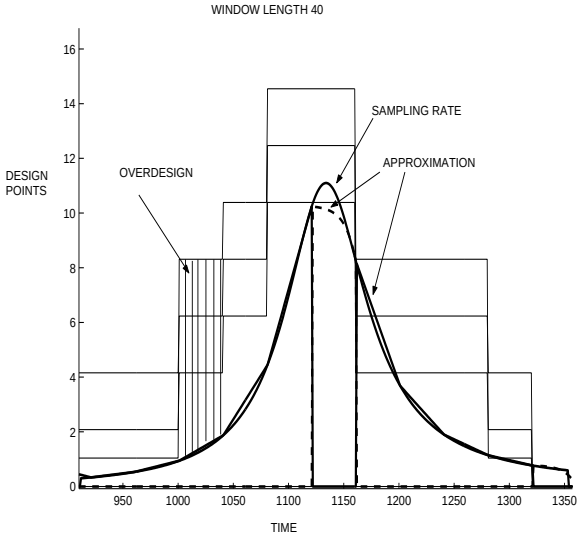

Figure 5.10: **DP selection for small time windows and fixed DP set**

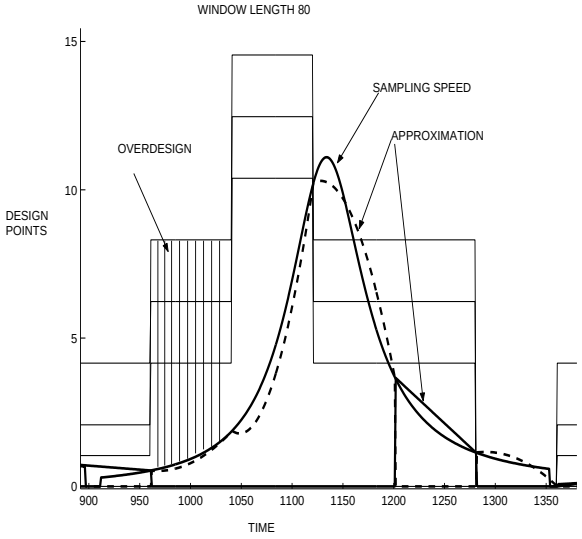

Figure 5.11: **DP selection for large time windows and fixed DP set**

Experiments show that having smaller time window length reduces ODF significantly. For example, for a fixed DP set, decreasing the window length from 200 seconds to 40 seconds reduced ODF by more than six times. The ODF reduction is about ten times for flexible DP selection. This explains that architecture adaptation is very beneficial especially for situations in which node processors execute cumbersome algorithms, e.g., image processing. ODF is about four times smaller for a flexible DP set compared to a fixed DP set. This motivates that it is beneficial to select first an optimal set of DP points, which is then used to design a customized image processor for that set of supply voltages. Energy consumption can be up to one order of magnitude smaller as compared to using a general-purpose processor with a pre-defined set of supply voltages.

5.4 Conclusion

The chapter dealt with furthering the modeling of the processing subsystems performance. The variation in the requirements of the processing subsystem. Although the section promotes with modeling the analytical functions which approximate the real performance curve over small windows the modeling technique was found to be somewhat cumbersome. The part of the modeling scheme which deals with discretizing the performance space and defining design points however was found to be useful in the next step of the research effort. The concepts of over design factors and optimal window lengths have been carried foward however characterising the performance requirements with analytical functions wasn't pursued further. The modeling with queues and random processes was used in the subsequent efforts.

Chapter 6

Energy Conscious Online Architecture Adaptation for Varying Latency Constraints in Sensor Network Applications

6.1 Introduction

Sensor network applications face continuously changing environments, which impose varying processing loads on the sensor node. This paper presents an online control method which adapts the architecture to minimize energy consumption while satisfying varying latency constraints. The method predicts processing load requirements over a finite time window and accordingly adapts the architecture. The behaviour of the hardware modules over time has been approximated with a Continuous Time Markov Process. Adaptive image processing for vehicle tracking was used as a case study for this approach. Sensor networks are emerging as a main technology for many applications in national security, health care, environmental monitoring, infrastructure security, food safety, manufacturing automation and many more [60] [77]. In fact, the vision is that sensor networks will offer ubiquitous interfacing between the physical environment and centralized databases and computing facilities [109]. Efficient interfacing has to be provided over long periods of time and for a variety of environment conditions, like moving objects, temperature, weather, available energy resources and so on. In this context, a key problem that ought to be tackled is that of devising embedded software and hardware architectures that can effectively operate in continuously changing, hard-to-predict conditions. In addition, architectures should be cheap and consume tiny amounts of energy - considering that their batteries are hard to replac or replenish.

For moving vehicle tracking, one of the main applications of sensor networks, the vehicle's velocity, trajectory and position defines the required sampling rate, hence the latency requirement for image processing [109]. In this

case, architecture adaptation is challenging because static, off-line prediction of a vehicle's movement is quite inaccurate in real-life. Even if vehicle movement is predictable, the resulting off-line model is highly non-linear and discontinuous in many points. Therefor, it is quite inefficient to address this architecture adaptation problem by using typical embedded design methods [6] [21] [46] [76]. These consider static, quasi static or stationary scenarios, which all can be described through fixed, off-line models. As explained in the paper, vehicle tracking requires on-line model identification as well as continuous architecture adaptation to varying performance needs.

This paper presents a novel approach for online customization of embedded architectures that function in non-stationary environments. The crux of the approach is a synthesis technique for developing online controllers that adapt the data-path of an architecture to varying latency constraints while minimizing energy consumption. The approach includes three steps: *(i)* look ahead on performance parameters (like image sampling rate and system latency) by buffering input data coming in a given time window, *(ii)* dynamic processing requirements prediction using a linear estimator activated at the end of every window period, and *(iii)* on-line architecture adaptation. Since our design is for a non-stationary environments, the control policy varies with the environment but is stationary within a time window. Adaptive control policy design is based on expressing the operation over time of the data-path blocks as Continuous Time Markov Process (CTMP). A set of linear equations is set-up to reflect block utilization rates, buffer space constraints, and total energy consumption. Obtained utilization rates affect the adaptation thresholds of control policies. For systems with high utilization we could achieve upto 29% lesser power compared to greedy policy.

The proposed architecture adaptation method is different from other dynamic adaptation techniques including on-line software optimization and runtime hardware reconfiguration [33] [56] [113]. For example, dynamic partitioning for reconfigurable hardware selects the regions of the binary or source code to be sent to hardware modules [85] [86]. Other approaches are for reducing energy consumption by dynamic resource allocation [46] [64]. They predetermine the hardware configurations for certain static design requirements, and then search among them by using heuristics to improve energy consumption. We view the problem in a slightly different manner and feel there is a need to have a quick though sub-optimal control methodology for systems functioning in drastically varying dynamic environments, in which, optimal static policies do not exist, or exist only in the case in which under utilization of powered up hardware resources is ignored. In the latter case, the system suffers drastic overdesign to meet performance constraints under all possible load conditions.

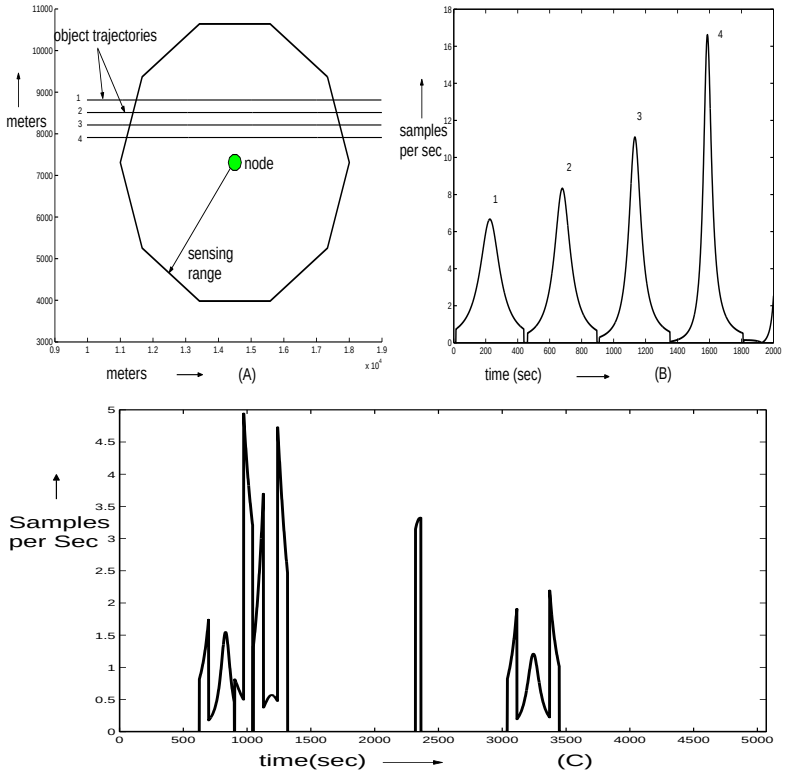

Figure 6.1: **Trajectories and Sampling speeds**

The paper is organized as follows. Section 2 presents a motivating example and Section 3 defines the adaptive control problem. Section 4 details the mathematical modeling of the system and of the control policy. Section 5 is a discussion of results followed by conclusions in Section 6.

6.2 Motivating Example

In order to strengthen our case for the presence of highly varying process load environments, we present an example in which we show a relatively simple moving objects tracking scenario. We considered a camera based sensor [109], which is tracking a moving object, such as a person or vehicle. The tracking granularity requirement demands one image sample per meter of distance traveled by the object, in order to have a trace of the object's trajectory accurate to within one meter distance of the object's actual location at all times. If the object is traveling at a speed of 20 m/s this sampling speed would translate to having 20 samples/s for the camera.

Obviously, varying velocity of the object would require variation in the sampling speed, but the distance of the object from the sensor as well as the angle at which the object is traveling with respect to the focal plane could cause additional variation in the sampling speed requirements, as shown in Figure 6.1. In Figure 6.1(a), though trajectories are straight, the velocity component tangential to the camera's focal plane is varying, but varying smoothly. The sampling speed requirement changes faster for the trajectories nearer to the sensor, as observed in Figure 6.1(b). It may be possible in this case to create estimation models for the sampling rate, but this is a simple and non realistic scenario as compared to the one in Figure 6.1(c). Sampling rates in Figure 6.1(c) do not follow a smooth variation, and include points of discontinuity due to sudden changes in the vehicle's movement, like stopping, changes in direction, acceleration, and so on.

Through the mentioned example, we identified following attributes for sampling speed (thus also system latency) variation during vehicle tracking:

- Sampling speed and latency constraints are constantly changing without following any particular mathematical law. Performance variations might include several peak and bottom points and different convexities, concavities and discontinuities.
- Performance requirement magnitudes pertain to broad ranges of values. For example, the latency requirement for a busy sampling period can be $10\times$ or even $100\times$ higher than the sampling need for idle times.
- Performance variation gradients are in a wide range. Figure 6.1(a) shows that some variations are quite mild whereas others are very steep.

For this type of applications, it is difficult to formulate a static mathematical model that estimates performance needs without resulting in gross mispredictions. Such a "hypothetical" model would be highly nonlinear, discontinuous and partially defined. Most of the existing embedded design methods [21] [46] cannot be used in this case, as they need well defined, static description of performance constraints. It is intuitively understood that statically calculated optimizations are of little relevance in cases not covered by the estimation models. Stationary optimal control policies are quite unsuitable [24] because of discontinuities that are difficult to be handled by ordinary differential calculus. Hence, due to the discontinuous nature of the performance curves, these systems fit better into the framework of discrete events [7][54]. We modeled the system dynamics with discrete events formulated over a fixed window of time used to sample the future performance requirements of the system. As it is also difficult to predict the possible changes in process-

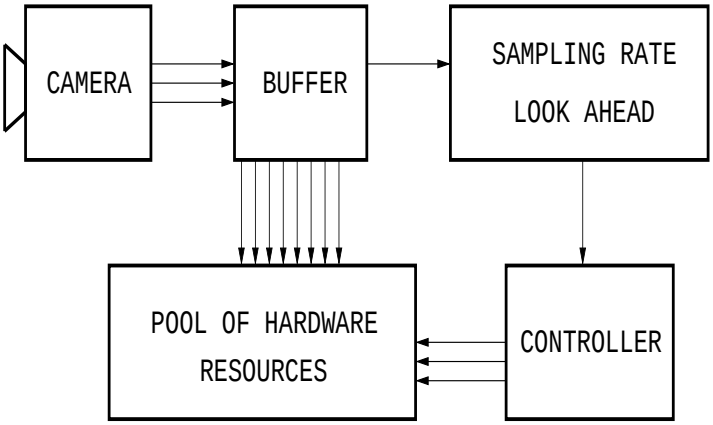

Figure 6.2: **Architecture for proposed methodology**

ing load, a finite look ahead on this processing parameter is the best way of learning future processing loads.

6.3 Problem Description

Problem Definition: For a given hardware architecture, device an adaptive, on-line control policy for each hardware resource such that (a) fixed buffer space and (b) varying latency constraints are met and (c) energy consumption is minimized.

To address the specifics of moving vehicle tracking problems, we propose a processing approach based on following three defining points:

1. There is a look ahead on performance parameters like sampling rate and latency.
2. Based on the look ahead there is a dynamic processing requirement prediction.
3. Online architectural adaptation takes place to reduce energy consumption and meet buffer space and varying latency constraints.

Look ahead and dynamic performance prediction is conducted within a fixed *time window* over which the data collected is to be processed.

Figure 6.2 presents the architecture that we used for implementing dynamic adaptation. During look ahead for the next window, the incoming data is buffered. The controller uses the inputs from the sampling rate look ahead to update its control policy. The controller makes changes to the pool of hardware with the updated policy after each window.

The time *window length* (WL)is a design parameter, and will have to be decided by the designer based on empirical data obtained from simulations of the particular application. Intuitvely, large time windows allow superior prediction but lengthen the adaptive response time and need more storage space. We ran several experiments to view the possible trends in the processing load variation as well as looked at the application's specification to gauge what window size would serve us. Although smaller window sizes would track the variations in requirements better, we have a lower limit to the window length. Specifically, we selected the limit to be the time required to process one sample if all resources of each type were used.

We defined *data load* (DL) as the data amount in terms of number of samples, which must be procesed in a time window. In the tracking example this amount of data may vary, depending on the rate the object moves. If DL is high then the number of hardware resources turned on is larger in order to meet the tighter timing constraint. There is an upper limit to DL based on the the hardware processing blocks and memory space being made available for it by an architecture.

6.4 Mathematical Modeling

We characterized the dynamics of the system architecture in the following manner. The state of the sytem is defined in terms of the resources that it is made up of. For example, a system with L, M and N number of resources of types R^1, R^2 and R^3 would have a state space vector S of the form $S = \{R^1_1, R^1_2 ... R^1_L, R^2_1, R^2_2 ... R^2_M, R^3, R^3_2 ... R^3_N\}$. Hence, if there are 10 elements for each of three types of resources, which are, ALUs, shifters and multipliers, the possible values that could be taken by each element of the state space is a cost which depends on the status of that element. In addition, each resource type R^i can be in one of its Z different *modes*. Resource modes depend on the processing activity of a resource and its present power mode status. Every element of each resource type has following four modes: (1) powered up and processing, (2) powered up and idle, (3) powered down, and (4) powered down and being requested. Please note that there are only two control actions asociated with each element, power up and power down. The obtained control policy is used to implement the controller block in Figure 6.2.

We encountered two major decision making steps while doing the mathematical modeling of this problem. The first being what mathematical framework would best model the variation of the system state over time, and the second being what control policy could quickly adapt with the system. The

traditional methods [6] [46] [76] have been designed for static or stationary environments where there isn't a need for adaptation in the control policies. These issues were presented next.

A. System state variation modeling. As motivated in Section 2, we decided to express the system state variation using difference equations. The difference equation that characterises the system is given by

$$S_{k+1} = f\{S_k, U_k, E_k\} \tag{6.1}$$

which states that the next state is a function of the current state, the control vector U_k and certain look ahead E_k. The control vector U_k is a set of control actions that were taken at the k_{th} time instant. The sampling rate variation has been obtained by monitoring the incoming data rate, which works at the granularity of the window length. The effect of this look ahead is modeled by the term E_k.

Buffering data over the time window allows the control method to obtain knowledge of the latency requirements of the system for the window period, hence it can incorporate this knowledge into the decisons it makes.

B. Control policy. The controller implements the following control policy, which consists of certain state transitions under certain conditions. The decisions taken in the control policy are which hardware elements and how many of them to turn on or turn off. The control policy scales in the following manner for all types of elements, though the actual policy for each will differ in numbers.

$$U_{(w)} = \begin{cases} N_4^T, N_2^T, N_3^T \longrightarrow N_1 & if\ SR_w < SR_{w+1}, \\ N_2^T \longrightarrow N_3\ and\ N_4^T \longrightarrow N_1 & if\ SR_w = SR_{w+1}, \\ N_2^T, N_4^T, N_1^T \longrightarrow N_3 & if\ SR_w > SR_{w+1}, \end{cases} \tag{6.2}$$

SR_w and SR_{w+1} are the sampling requirements of the current window and the next window respectively. SR_{MAX} is the maximum sampling speed the system can tackle. Beyond this sampling speed there will not be enough hardware resources to speed up the processing. The number of elements that make the transition from current state i are given by N_i^T. N_j is the number of devices that need to be in next state j during next time window. The elements that transit to state j follow the priority with which they are listed. THe policy has been formulated in the following manner for the first case of the control policy shown above.

$if\ SR_w > SR_{w+1}$,

$$N_1 = \begin{cases} \sum_{u \in U_w} \lambda_{1,u}\ N(1 + \Delta SR_w) & if\ SR_{w+1} < SR_{MAX}, \\ N & otherwise, \end{cases} \tag{6.3}$$

$$N_2^T = \begin{cases} N_2 & if\ N_1 \geq N_2 \\ N_1 - N_2 & otherwise \end{cases} \tag{6.4}$$

$$N_4^T = \begin{cases} 0 & if\ N_1 < N_2 \\ N_4 & if\ N_1 - N_2^T \geq N_4 \\ N_1 - N_2 - N_4 & if\ N_1 - N_2^T < N_4 \end{cases} \tag{6.5}$$

$$N_3^T = \begin{cases} 0 & if\ N_1 - N_2 < N_4 \\ N_3 & if\ N_1 - N_2^T - N_4^T \geq N_3 \\ N_1 - N_2 - N_4 - N_3 & if\ N_1 - N_2^T - N_4^T < N_3 \end{cases} \tag{6.6}$$

Equations (3)-(6) basically state the optimal control policy according to which any element, which is idle and powered up should be either shut down or put to use depending on the next windows latency requirements. The prediction of the powered up elements in the next state N_1 was done by taking into consideration the change in sampling rate $\Delta SR = \frac{SR_w - SR_{w+1}}{SR_w}$, as well as the *likelihood* λ of an hardware element being in state "1" (power up and processing) which we have approximated by the steady state probability of an element being in state "1" $\alpha 1 = \sum_{u \in U_w} \alpha_{1,u}$. Linear equations (7)-(11) are solved to find the steady state probabilities$\alpha_{i,u}$. Likelihood captures the global influence of a certain hardware resource on system performance, thus it jointly reflects the criticality of the block with respect to timing, buffer size needs and energy consumption, as well as the amount of resources of that kind in an architecture.

The policy states that if any element is powered down and being requested it should be turned on or shut down again depending on sampling rates. The power down or power up has to be done from different states depending on the severity of the gradient. Thus, elements in state "4" should move to state "1" first, then if N_1 is not satisfied, elements from state "2" should make transitions, and so on. This is a set of constraints for the elements of type R^1 which are N_1 in number. Similar constraints exist for the other elements and for the two other cases of the control policy.

C. Finding the likelihood factors. Finding the precise likelihood value of a resource is an NP-complete problem, as it requires finding the optimal architecture for given constraints. Instead, we approximated likelihood with the steady state probability of a single hardware element modeled as Continuous Time Markov Chain (CTMC) [24], as shown in Figure 3. CTMC were previously used for control policy design, including power mode controllers [6] [76] and bus arbiters [42]. Another advantage of this modeling is that it offers - for each hardware resource, a figure of merit that cumulatively expresses its time criticality, usage, energy consumption and impact on buffer size. Defining likelihood using an heuristic cost function would have been an alternative. However, having no rigorous mathematical support, we avoided this possibility.

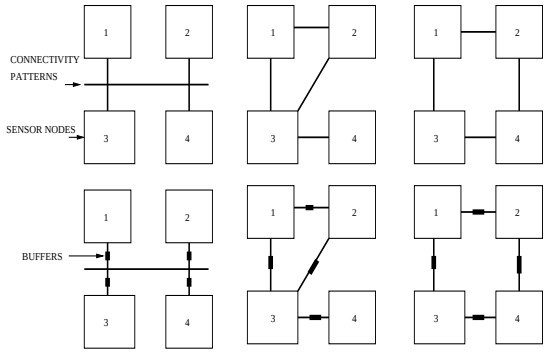

Figure 6.3: **State transitions for power modes**

*

Steady state probabilities $\alpha_{1,u}$ were calculated by modeling the mode changes $q(j,u)$ are the rates for choosing certain transition u while in a certain mode j. The rates $q(i,j,u)$ and $q(j,u)$ have been obtained by using a simple scheduling algorithm. The scheduling algorithm works on a threshold of time after which the hardware element powers down. The threshold depends on the energy consumed in powering down $E_{turnoff}$ $+E_{turnon}$ and the energy saved by being in the power down mode over a certain period of time $P_{dn}t$. The time threshold is given by T_{th}, $T_{th} = \frac{E_{turnoff}+E_{turnon}}{P_{dn}}$ Thus for all idle times $t_{idle} \geq T_{th}$ the hardware element will be powered down.

The rates $q(i,j,u)$ are the rate for selecting a certain action while being in state i and going to state j, as shown in [76][24]. $Bprop_k i,a$ are the buffer occupancy rates for the hardware elements, and we treat them as costs which cannot exceed a certain amount of available buffer space.

$$\sum_{u\in U(j)} q(j,u)\alpha_{j,u} - \sum_{i\in I}\sum_{u\in U(i)} q(i,j,u)\alpha_{i,u} = 0, j \in S, \tag{6.7}$$

$$\sum_{i\in S}\sum_{u\in U(i)} Bprop_k(i,u)\alpha_{i,u} \leq C_k, k = x,y,z, \tag{6.8}$$

$$\sum_{i\in S}\sum_{u\in U(i)} \alpha_{i,u} = 1, \tag{6.9}$$

$$\alpha_{i,u} \geq 0, i \in S, u \in U(i), \tag{6.10}$$

The goal of minimizing the energy consumption of the system while meeting buffer size constraints is expressed by the following set of equations.

$$minimize \;\; C_{window} = \sum_{i=1}^{N}\sum_{u=1}^{U(i)}\sum_{k=1}^{D} C_k\alpha_{i,u} \tag{6.11}$$

C_{window} is the total cost over the time window, and is the sum of the cost incurred at each clock cycle.

$$C_k = \sum_{a=1}^{L} Cx_{a,k} + \sum_{b=1}^{M} Cy_{b,k} + \sum_{c=1}^{N} Cz_{c,k} \tag{6.12}$$

The cost at k_{th} clock cycle is given by a summation of the costs incurred by all hardware elements. We would like to state at this point that this summation of costs is simply in the mathematical modeling, and the controller will not be performing such summing operations, and will not be looking at the cost function. These equations will be solved offline and the solution shall be encoded into the controllers. If each hardware processing block has the four modes presented in Section 3, the cost functions of the elements follow the following pattern.

$$Cx_{i,k} = \begin{cases} P_{up}\Delta t + B_{prop} & if\ x_{i,k} = 1, \\ P_{up}\Delta t + B_{prop} & if\ x_{i,k} = 2, \\ P_{dn}\Delta t + B_{prop} & if\ x_{i,k} = 3, \\ P_{dn}\Delta t + B_{prop} & if\ x_{i,k} = 4, \end{cases} \tag{6.13}$$

The cost of an element in the powered up states (state "1" and state "2") is the costs associated with being powered up P_{up}. In state "3" and state "4" the cost is simply that of powering down, and in each state there is an added penalty based on the buffer space that got occupied while the element was being requested B_{prop}.

The equations could be solved for the state space as a whole but this would lead to too many equations due to the large state space we are dealing with. We have made an assumption that the steady state transitions probabilities of an individual element would be the same for all elements of its type. Another assumption we made is that it is safe to consider the steady state transition probabilities of each element independent of the probabilities of the other, since the timing delays of the elements are different and obey the following relationship $Tmull >> Talu = Tshift$ and the occurrence of instructions that utilise these elements in an algorithm is different too. The occurrence rates of intructions which could use these resources in the RGB to CMY colour conversion algorithm that we considered had a the highest rate for intructions which could use ALUs followed by multipliers and shifters. In other words the number of adders powered up has no correlation with the number of multipliers powered up as they wouldn't make much difference in satisfaction of overall timing constraints.

D. Controller circuit. The output of the set of equations is the steady state probabilities for the state and control action pairs given by $\alpha_{i,a}$ as shown

in [24]. These steady state probabilities $\alpha_{i,a}$s with the sampling rate are used in the controller. They provide the basic structure for the control policy and these are then scaled by the sampling rate as in equation (3).

6.5 Experiments

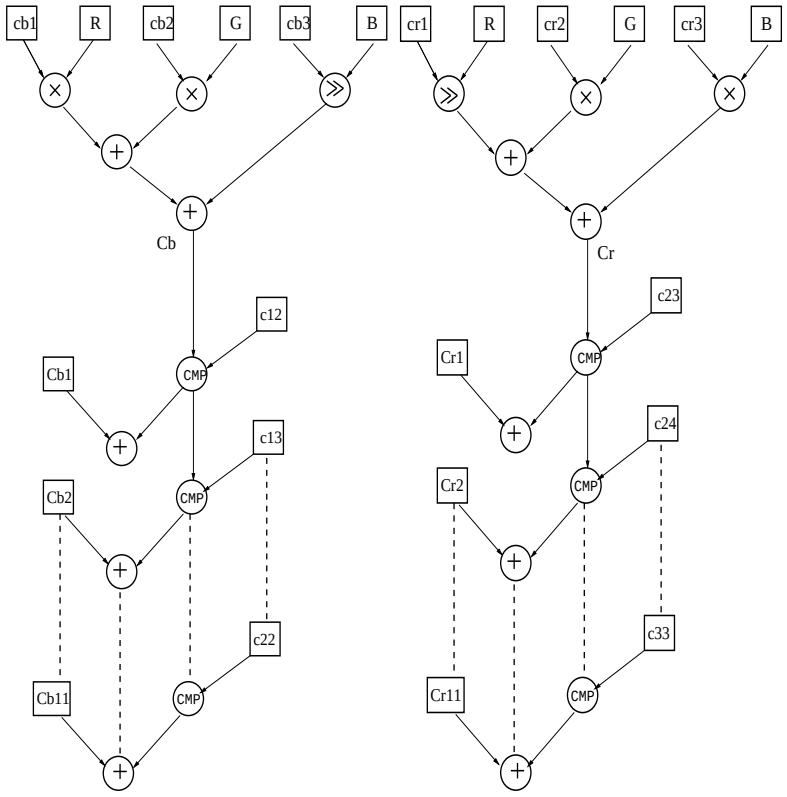

Figure 6.4: **Taskgraphs**

The task graphs we used have been shown in Figure 6.4. The task graphs used belong to the R,G,B to Y,Cb,Cr colour space conversion algorithm, which is composed of three task graphs executed in parallel [109]. Several instances of thistask graphs may also be run on parallel on separte pixels of the image. Though the graphs have similar structure their execution times will depend on the incoming data due to the several data dependent branching operations. The powering up and powering down of Hardware resources consequently has differing rates even if the processing load or samples per sec. remains the same.

We implemented a SystemC model of an architecture which has a bank of 10 ALUs, 10 Multipliers and 10 Shifters. Each of the resources has a controller which implements the control policy discussed in section 4. The elements are connected to a data bus, which carries the sampled data. The controllers are sensitive to the sampling rate and perform a buffering operation depending on *Data Load* and a look ahead on the sampling rate before powering up or powering down resources during a window. The equations (7)-(11) were solved using MATLAB6p1 to obtain steady state probabilities $\alpha_{i,u}$. The steady state probabilities for each type of resource were then embedded into the controllers.

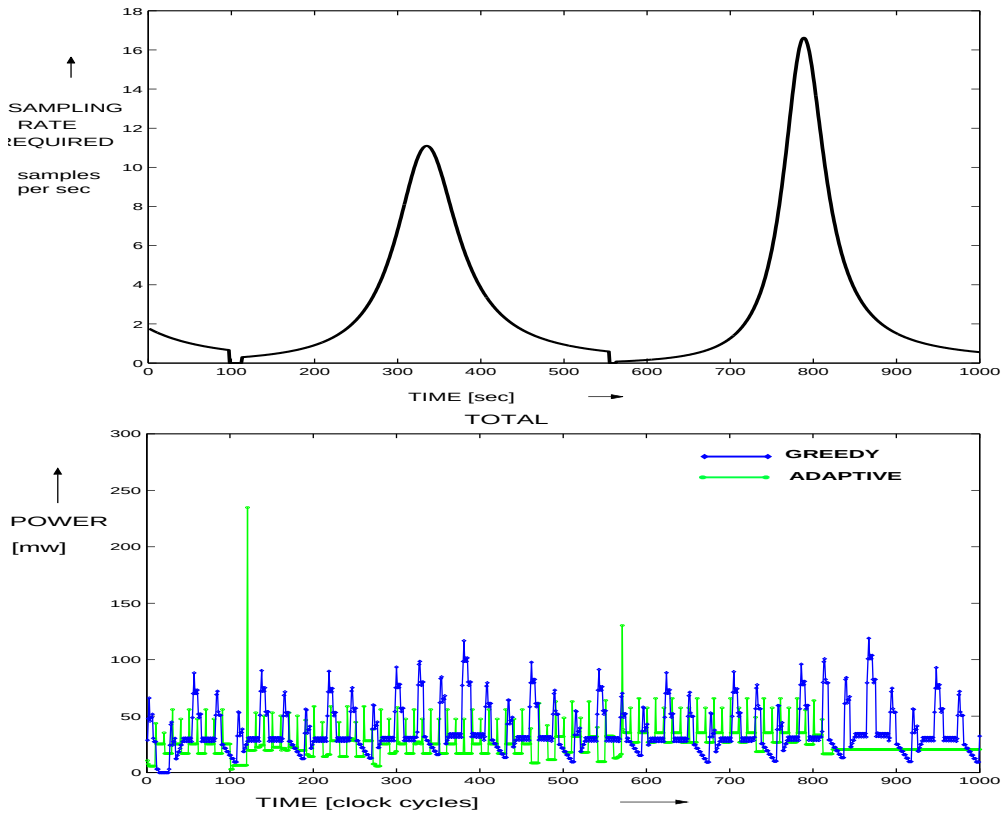

Figure 6.5: **Sampling Requirements and Total Power**

The sampling rate variation has been shown in Figure 6.5. The reason we chose such variation is we wanted to test how well the control policy adapts to the varying required processing rate. The power consumption trend follows the required sampling rate as more resources get turned on to meet the smaller latency constraints for higher sampling rates. The "Greedy" policy which turns on every requested resource and turns them off based on a time threshold T_{th},

to which we compared our "Adaptive" policy, tends turn on more resources hence has a larger power consumption. To its advantage the Greedy policy manages to track the variation in processing requirements faster. For a window length of 10 clock cycles and a *data load* of 480 we found the total power consumption in the adaptive policy was 29% lesser than that of the greedy policy. The *Data Load* of 480 signifies that a maximum of 480 samples could be buffered, though the actual maximum buffer space used was close to 200 samples hence we could have applied tighter buffer space constraints on the equations (7)-(11) and used a smaller buffer space.

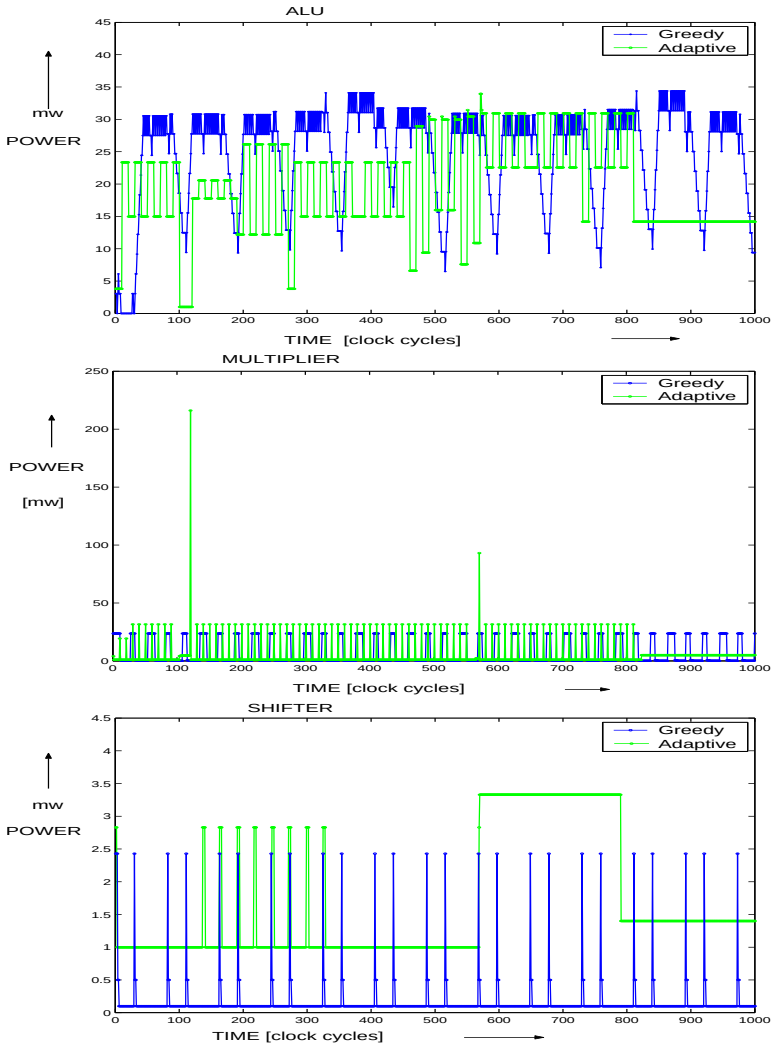

Figure 6.6: **Power trends for different resources**

The individual types of resources had their own controllers, hence had different trends in terms of their power consumption. The ALU had a high utilization rate due to the large number of ALU operations in the task graphs. Due to the large number ALU operations the ALU bank had fraction of ALUs powered on all the time and extra ALUs were turned on/off depending upon the requirements as shown in Figure 6.5. The greedy policy tracked the variation better but overdesigned in terms of turning on more ALUs which became idle over a certain fraction of the *window length*.

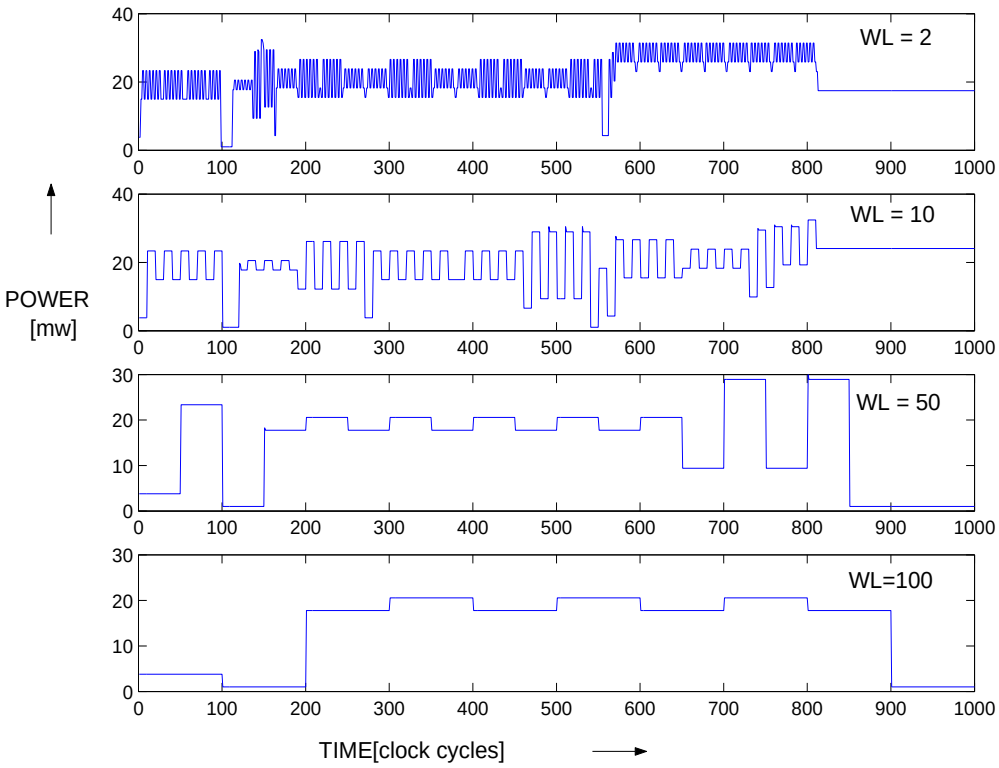

Figure 6.7: **Varying Window Lengths**

The shifters had a low utilization as there are only 2 shift operations among the 81 total operations in the three task graphs. The greedy policy turned on the shifters exactly when needed and performed better than the adaptive policy in terms of power consumption. The adaptive policy performed poorly as it tried to adapt when there really wasn't need to adapt. The adaptive policy kept some shifters powered up even though they were idle as seen in Figure 6.6. It also unnecessarily powered up extra shifters while trying to adapt to the second peak in processing requirements shown in Figure 6.5.

The multipliers had a low utilization rate similar to the shifters and the

greedy policy performed better in terms of lower power consumption. We see a large spike in the multipliers graph in Figure 6.6. This is because the adaptive policy tried to turn off all multipliers when the sampling rate went very low as shown in Figure 6.6. This created a backlog of "multiply" operations which were then serviced at once by turning on extra multipliers.

The window length has to be carefully selected in order to allow the adaptive policy to track the variation in the sampling requirements yet conserve power. This effect can be observed for the ALU banks from Figure 6.7, in which the adaptive policy tends to look more like the greedy policy and consumes more power while trying to track varying processing requirements, for example, when window length (WL)=2. With longer window lengths (WL)=10, 50 and 100 the policy reduces power consumption while being insensitive to the varying latency requirements, thus the adaptive policy will require larger buffer space for larger windows. In case the window length is large its possible that the hardware resource will become idle over a fraction of the window length thus wasting power. Since the ALU had a high utilization rate this did not occur. In resources with low utilization the power trend with varying window size was different.

In the experiments the power savings from the ALU dominate the poor performance of the control policy in the case of shifters and multipliers. In case of low utilisation rates of resources or low occurrence rates of instructions which could utilise a particular resource the greedy policy, which turns on all requested resources, may perform comparably with the adaptive policy. In future attempts we shall apply the adaptive policy to a commercial core which has power down modes and selectively controllable resources.

6.6 Conclusions

We have shown that the adaptive policy performs well for resources with high utilization rates under varying latency constraints. We could obtain up to 29% lesser power consumption compared to a greedy policy for certain design constraints. With well chosen window length and data load parameters we can get a control policy which is obtained offline, yet can be near optimal for online adaptation of embedded systems suffering varying latency constraints.

Chapter 7

Future Work

7.1 Continuous Time adaptation in the Memory subsystem

In modern day embedded system memories are hierarchical and consist of levels of successively faster and smaller memories closer to the processor. This is based on the data locality principle where in the data or a set of instructions required by the processor are close to each other. If the data or instructions are adjacent or near in the memory space it makes sense to move that sector of memory to a fast small memory close to the processor so that the memory access is faster. A description of the hierarchical memory subsystem has been given in [36]. In the hierarchical structure of memories has been shown. They are usually made up of a cache which is usually a fast small memory which contains some of the memory blocs that reside in the main memory. The main memory gets loaded with data from the disk. There are numerous trade offs between block sizes ,cache sizes and replacement policies which have been illustrated in [36]. We would like to extend the possible improvements to these methods by introducing continuous architectural reconfiguration to the hierarchical memory subsytem.

Reconfigurable memories with variable granularites of access have been proposed for different applications with very different memory access patterns [59]. The memory is modular with reconfigurable tiles and has been mapped to applications like the Hydra Multiprocessor[32] which has irregular and intermittent memory access as well as the Imagine Streaming Processor [78] which has a very regular memory access pattern due to the streaming media applications. The use software based statistical profiling data shall be used in the future to generate empirical distributions for the service requests being made. Instead of using closed form mathematical random process generators

to provide the service requests. The modeling methodolgy shall be much faster than RTL yet accurate like cycle count accurate modeling. The use of continuous time modeling has been used in some power management policies for DRAMs in which the gap between the memory access was modelled by an exponential distribution [22]. The modeling shall exploit the different power modes and configuration of the DRAM memory module as well as transactions at a chipset level.

We would like to explore using the constraints and trade offs involved in designing hierarchical memories and cache management to be extended to controlling the reconfigurable memory modules in hardware while being optimised for reducing miss rate, miss penalty and improving hit rate in a cache based hierarchical system. We shall attempt to throw the problem as an MDP problem too.

7.2 New Online Adaptation Methods for Constrained Reconfiguration

The predictive methods used in [74] [75] need offline solving for the generation of control policies. There are a new set of methods that provide online control. The manner in which these methods work is that they provide the system with a set of sstates which could be viewed as the goal states of the system. The system is working in order if it can obtain these states. The difference with regular dynamic programming methods which essentially work on similar paradigms of attaining an optimal state in finite time [25] is that the goal states are continuously updated depending on the physical stimuli provided as input to the controller. This method of controlling based on reaction to stimuli is called Reactive Planning which is an NP hard AI problem. It has been thrown in a Partially Observable Markov Decision Process (POMDP) framework in order to maintain the markov nature between the temporal variation of the states. The sytem model is a Hidden Markov Model which predicts the next possible set of system states with some probability. The control actions are chosen such that the probablity of the system being in a high reward goal state is maximised as in the predictive methods. These methods fit better into the continuous adaptation idea as they react to stimuli continuously and do not offline solution of equations in oredr to generate control policies.

7.3 New Performance Modeling and Simulator Building Techniques

RTL and cycle accurate simulators although accurate and necessary are extremely time consuming. Designers and testers do not need the accuracy provided by such methods at all times. Small and incremental changes in design can be analyzed by statistical methods faster and more efficiently as compared to resimulating at cycle accurate or RTL levels. Previous efforts at improvng statistical performance analysis have involved extracting important parameters on which a computers performance depends [118]. Most benchmark based performance analysis methods depend on sampling schemes and benchmark characteristics to speed up simulations [114]. Another approach is to save the state of a computer system and advance through a set of instructions in a functional fashion at high speed and then return to cycle accurate simulation [106]. These efforts are usually for certain subset of an instruction set [106].

This effort focuses on making statistical modules for the hardware. An entire piece of hardware would be replaced by a statistical module in a HDL or C++ simulator. This style of hybrid modeling would be an easy fit into transaction level models. Statistical simulators are useful in situations where the time taken to simulate the system at RTL level or even transactional or cycle accurate levels would taketoo long. The statistical characteristics of the hardware module will be hard coded into the module and would abstracted away from the effects of the individual benchmark. The statistical module shall be independent of *warm up* behaviour too and shall behave in the same fashion throughout the simulation.

The use of distributions allows us to analyze average cases as well as extreme cases and also gives us an idea how often a extreme case may occur. This helps prevent over-designing. Distributions could be closed form distributions that are generated from mathematical formulas or could be empirical distributions which have been obtained from raw data from a previous cycle accurate or transactional simulation which may be too painstaking.

The use of statistical mapping to model the architectures allows us to come up with fast closed form mathematical performance prediction methods. In some cases these methods can provide estimates in a few minutes.The use of models shown in Figure 7.1 where a statistical matrix is inserted into a cycle count accurate (transactional) model written in systemC or C++ would be particularly useful in the industry where the timing numbers for a particlar component or core may be unknown or uncertain. This matrix would convert

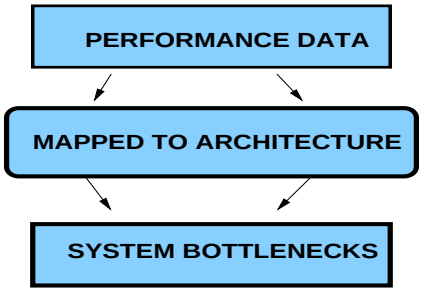

Figure 7.1: **Method**
software performance data into hardware bottleneck predictions.

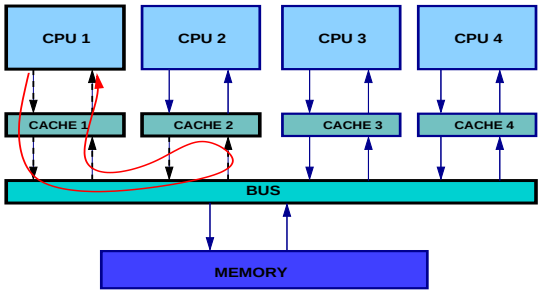

Figure 7.2: **snoop path**

Examples: The following are examples this effort is currently concentrating on.

- *Platform Transactions:* At a platform level there can be several cores or processors interacting with a shared bus as shown in 7.2 In such systems high level performance parameters are used to determine where the bottlenecks are. Throughputs, latencies and bandwidhts of buses and memory subsystems are checked to ensure the system meets performance requirements.

  The operation of the system can be modeled at a transaction level where different transactions such as cache access,snoops and memory accesses are modeled in terms of discrete number of clock cycles and probability distributions. The Figures 7.2 and 7.3 show the paths of a snoop and a memory read. These transactions could take differing amounts of time in clock cycles depending on hardware design-point parameters like queue sizes, cache sizes, cache sizes, bus availability and DRAM latencies. By creating distributions along these dimensions of parameters with respect to clock cycles valuable correlations could be obtained between the performance parameters and hardware design-points.

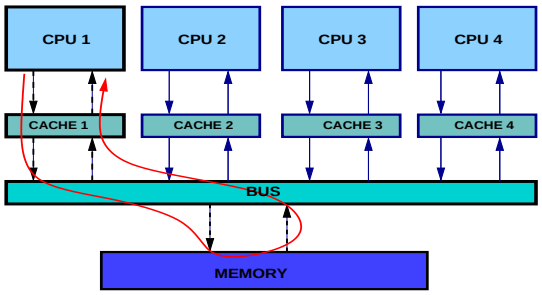

Figure 7.3: **read miss path**

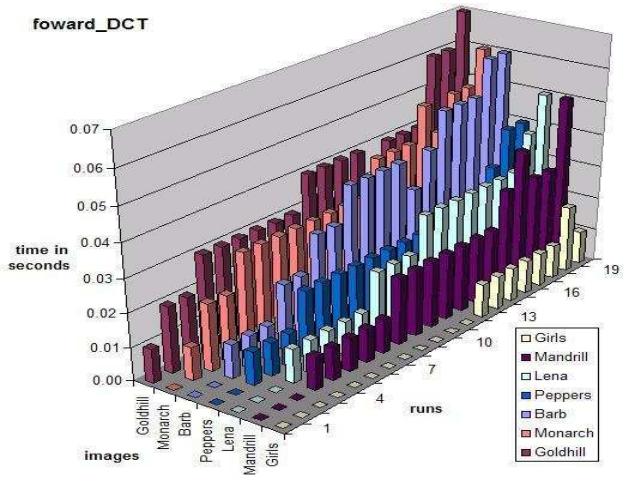

Figure 7.4: **execution times**

- *DCT module:*This work is being extended to incorporate software profiling data into the statistical models. The data shown in Figure 7.4is the execution time of a DCT method in a JPEG software run on a single processor. This data corresponds the input data as shown in Figure 7.1. use statistics from profiling on a single processor to estimate performance in case the same application is mapped to several processors. The attempts shall be to characterize relationships between the the timing distributions on a single procesor with those of the same application mapped to several cores in a NoC or multi core processors. In Table 7.3 the statistical module to replace a DCT hardware module has been shown. The designer has to plug in the statistical module in order to produce the waiting times in column 1 with the required probability distribution shown in column 2.

This is a very rough of approximation of the possible execution time distribution. Extensions could be to utilize the the percentage time execution data

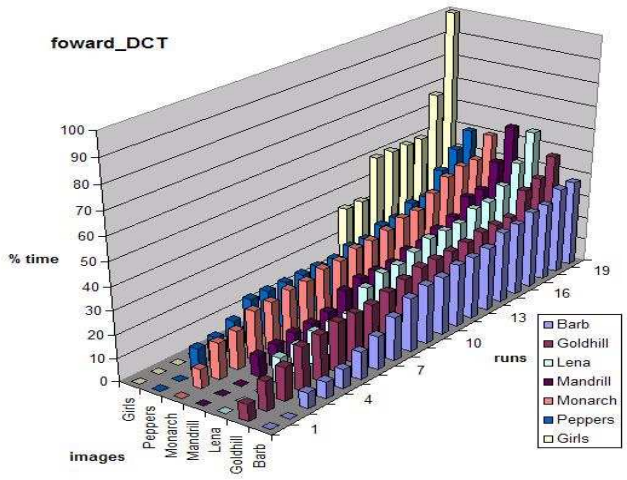

Figure 7.5: **percentage time**

Table 7.1: **Statistical Plugin**

EXEC. TIME VALUE seconds	EXEC TIME DISTRIBUTION
0.1	0.9
0.2	0.24
0.3	0.29
0.4	0.20
0.5	0.8
0.6	0.6
0.7	0.4

shown in Figure 7.5 to further skew the data to get more realistic distributions.

7.4 Summary

In this chapter we presented several possibilities for inprovement and extrapolation of our research efforts in contunuous adaptation of embedded systems. With this chapter we conclude our discussion and the next chapter is a set of conclusions. Statistical plug ins will speed up design and offer a framework to predict bottlenecks.

Chapter 8

Conclusion

8.1 Introduction

In conclusion we would like to present a brief summary about our goals. The ones we have achieved and the ones we wish to explore further. We also discuss the the methods and techniques used to achieve these goals.

The primary goal of the report was to present a discussion on the need and the ways of providing continuous time adaptation in the architecturtes of modern embedded systems. This involved a systematic study of various architectures and applications in which we could exploit the use of continuous time architectural reconfiguration. We also compared different control techniques in order to understand which would be most feasible in a continuously varying environment. We compared the control policies in order to ascertain their suitability by means of simulation. In the previous sections we saw the need for continuous time adaptation, the different control methods could be used in continuous time adaptation and experiments that support the use of stochastic models and stochastic optimisation to generate control policies for continuous time adaptation. A categorical enumeration of our acheivements has been listed below.

- *Using Stochastic Processes to Model On-Chip Traffic*: The use of stochastic models has been heavily used in reliability and fault tolerance testing of embedded systems [40] [82] [47] and used in some power management and estimation techniques [75] [74] [6] [65]. We have presented a strong case for extending this concept of stochastic modeling for SoC design, especially in the design of the control policies for varied subsystems. We have successfully used stochastic models of the communication subsystem in order to device arbitration policies for the communication subsystem.

- *New Online Adaptation Methods for Constrained Reconfiguration*: The predictive control methods need a precharacterisation of the workload which may be tedious and difficult to obtain. In [110] and [111] we see that control policies based on reactive systems have been designed. These would need a continuous update of the state as well as a set of *goal states* which are a set of states that would define the optimal operation of the embedded system. The systems are called reactive because the constraints change due the interactions within the states and with external stimuli causing a different set of *goal states* to be formed.

- *Continuous Time Adaptation*: The central theme of our effort has been continuous time modeling and we have successfully used continuous time models in order to capture the timing charactersitics of a modern day embedded system. We have provided a detailed study of the possible use of continuous time adaptation in various applications and architectures. The incentives for using continuous time adaptation are 1) not having to redesign systems from scratch in presence of changes in protocols or traffic patterns ,2) allowing the embedded systems to be intelligent and change themselves in a continuously varying environment and 3) Efficiently managing architectural resources while handling trade offs with respect to QoS and Power Consumption in a continuously varying environment.

- *Simulation of Queueing Models* : We have successfully used queues for modeling the bus architectures of real world applications. Though not novel ,the use of queueing theory to model the on chip communication helped us analyse the diferent arbitration policies for different architectures bye simulating the the traffic under control of a given policy. We checked the performance of the policies by sequentially running them on the queueing models and plotting their performance. We also extensicvely studied queues to understand various instances of micro-architecture where the architecture could be modeled by queues.

- *Using CTMDPs to generate control policies*: The use of CTMDPs allowed us to present the arbitration problem of the communication sub system as well as the hardware resource allocation problem of the processing subsytem in a formal mathematical framework. The policies generated using this method can be acheive theoretical optimality in terms providing a set of actions for maximising the possible reward obtained from any state of the system. Heuristic control policies do not

lend themselves well to an analysis of their optimality but have been compared with our methods by means of simulation.

The mapping of arbitration problem to the queueing network and further mapping the quantities from the queuing model to the linear programming based CTMDP solution was the novel and major contribution of the thesis. In the future we shall be extending continuous time architectural adaptation to the Memory Subsystem and transaction level models while improving the current models for the service request generators. We would also like to compare offline policy generation methods with online policy generation or AI based methods in order to see if the online or AI based methods can help us adapt the architecture better while in continuously varying environment. A major thrust shall however be made towards extending these methods towards platform and chipset performance testing where the fast performance prediction is imperative and even cycle count accurate simulations may seem too slow.

Chapter 9

Appendix 1

Results for experiments with varied traffic patterns on the Network Processor architecture.

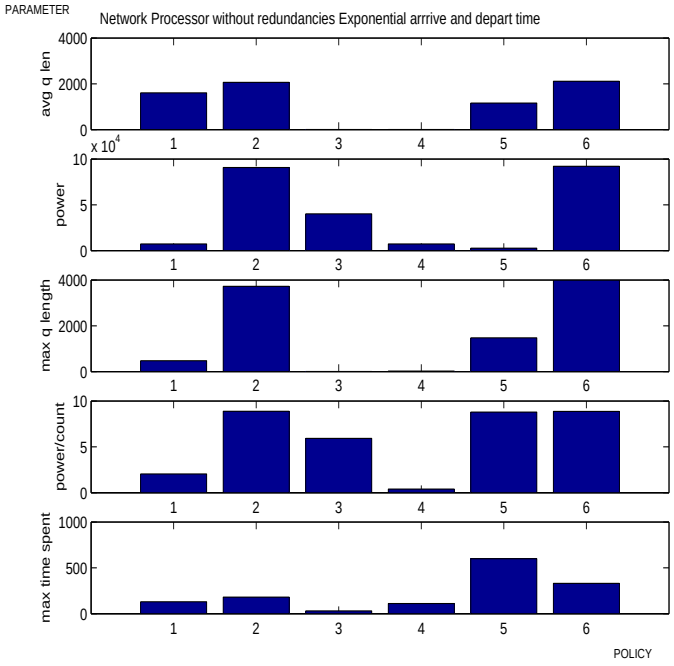

Figure 9.1: **Results for Network Processor without redundant paths exponential arrival and exponential departure times**

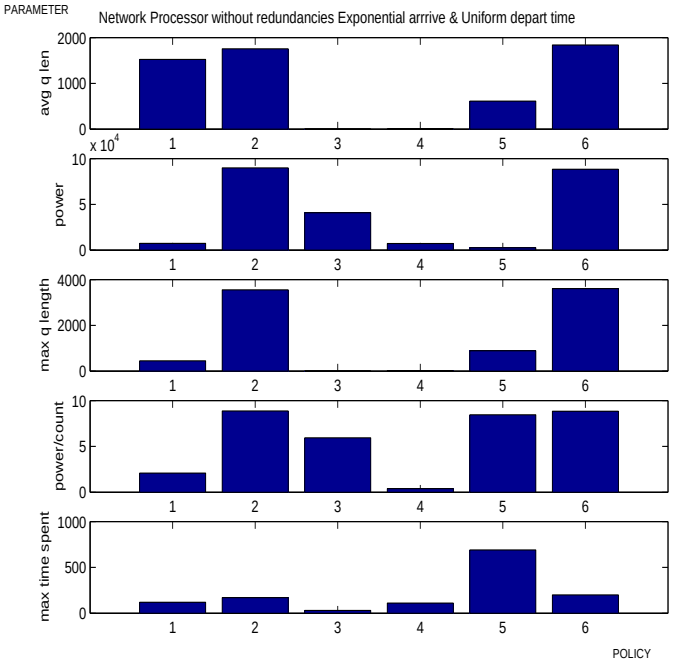

Figure 9.2: **Results for Network Processor without redundant paths exponential arrival and uniform departure times**

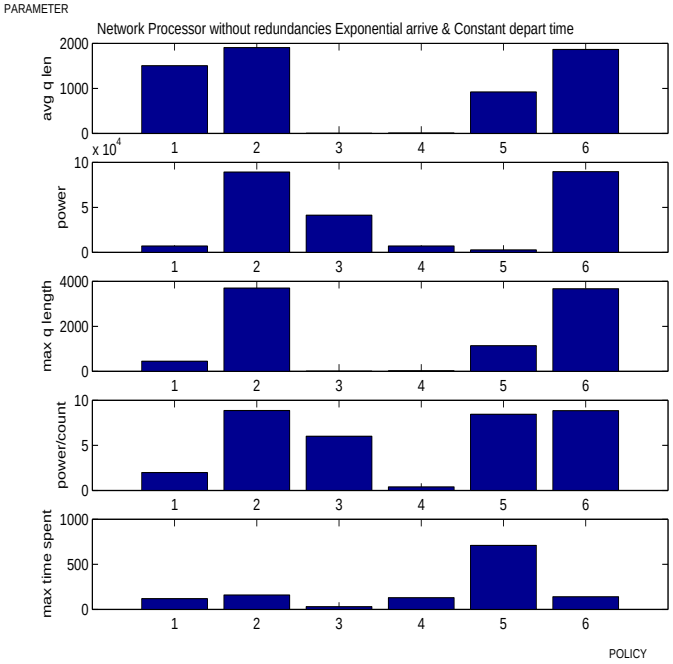

Figure 9.3: **Results for Network Processor without redundant paths exponential arrival and constant departure times**

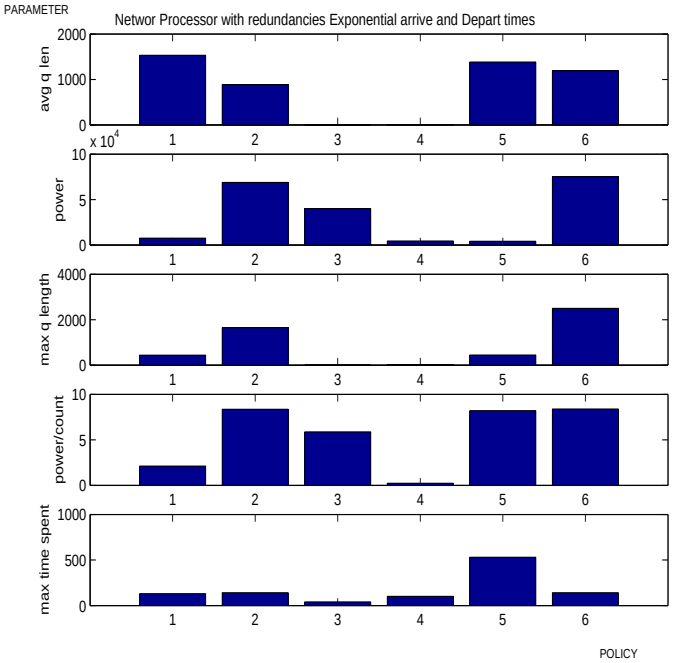

Figure 9.4: **Results for Network Processor with redundant paths exponential arrival and exponential departure times**

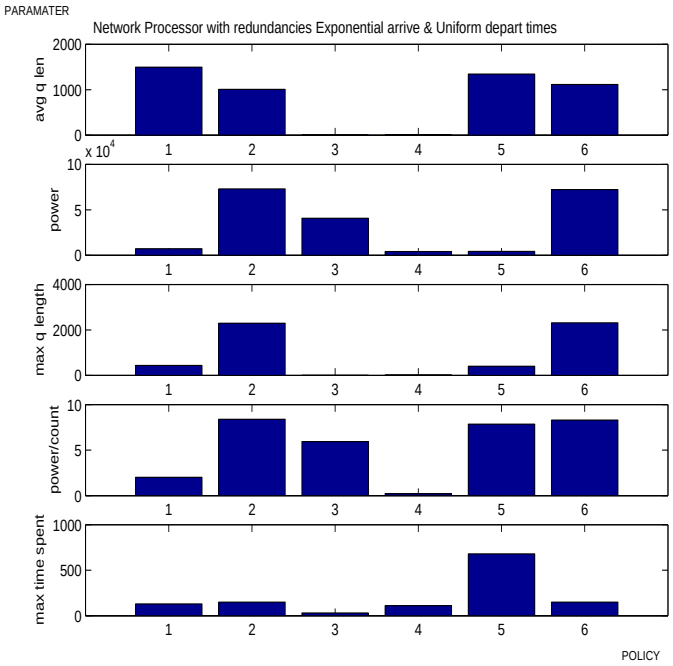

Figure 9.5: **Results for Network Processor with redundant paths exponential arrival and uniform departure times**

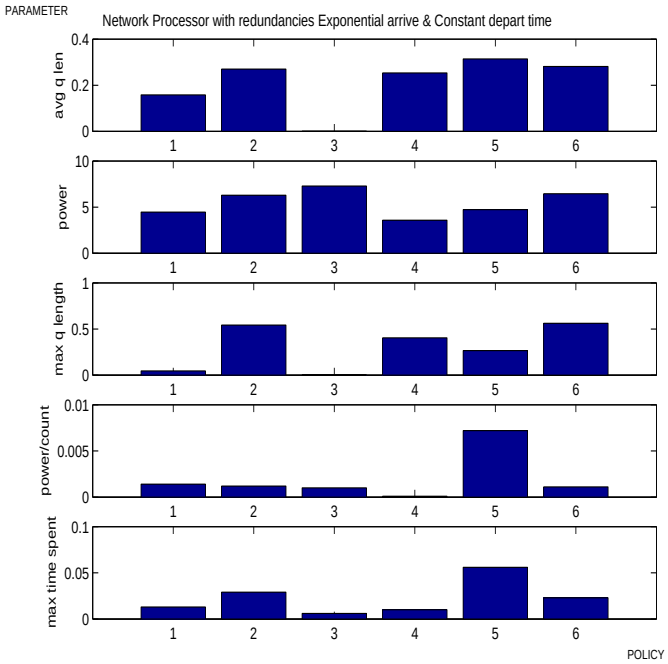

Figure 9.6: **Results for Network Processor with redundant paths exponential arrival and constant departure times**

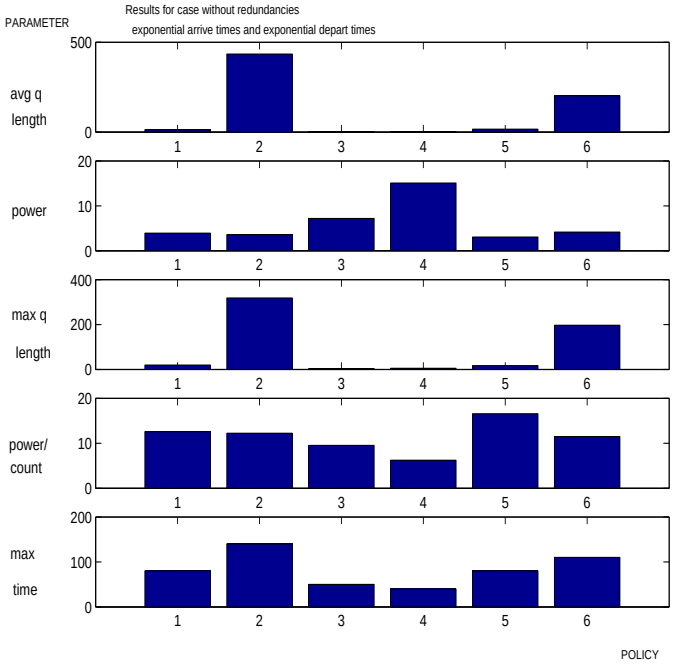

Figure 9.7: **Results for Network processor without redundant paths**

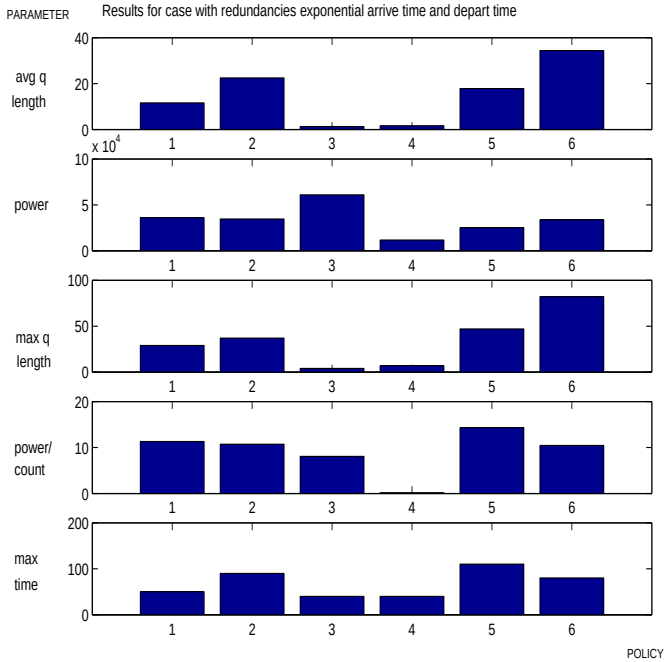

Figure 9.8: **Results for architecture in Figure 4.7 with redundant paths exponential arrival and departure times**

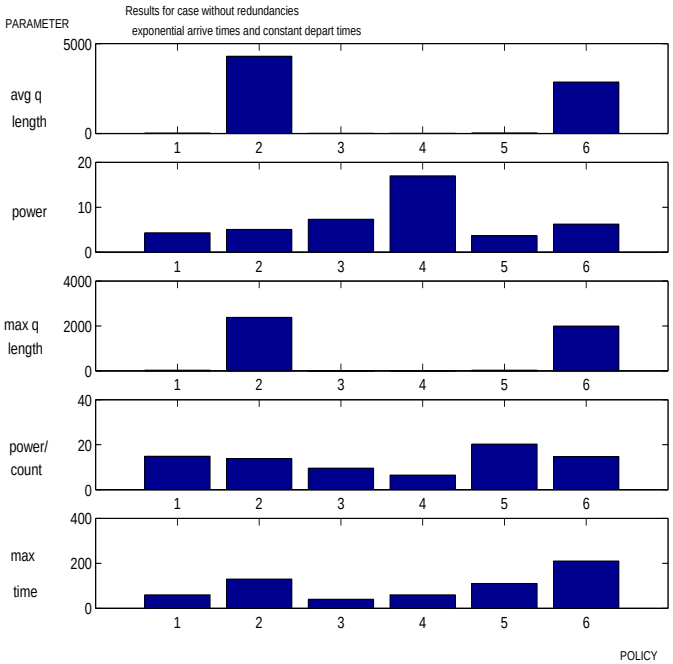

Figure 9.9: **Results for architecture in Figure 4.7 without redundant paths exponential arrival and constant departure times**

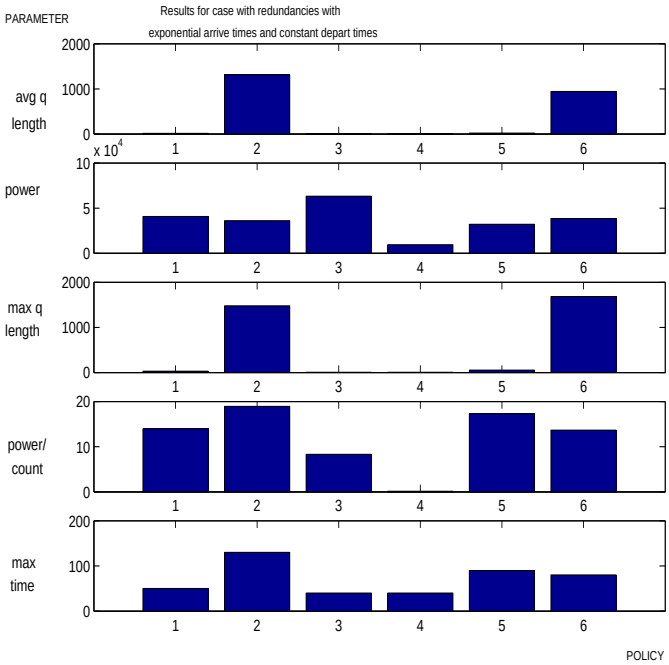

Figure 9.10: **Results for architecture in Figure 4.7 with redundant paths exponential arrival and constant departure times**

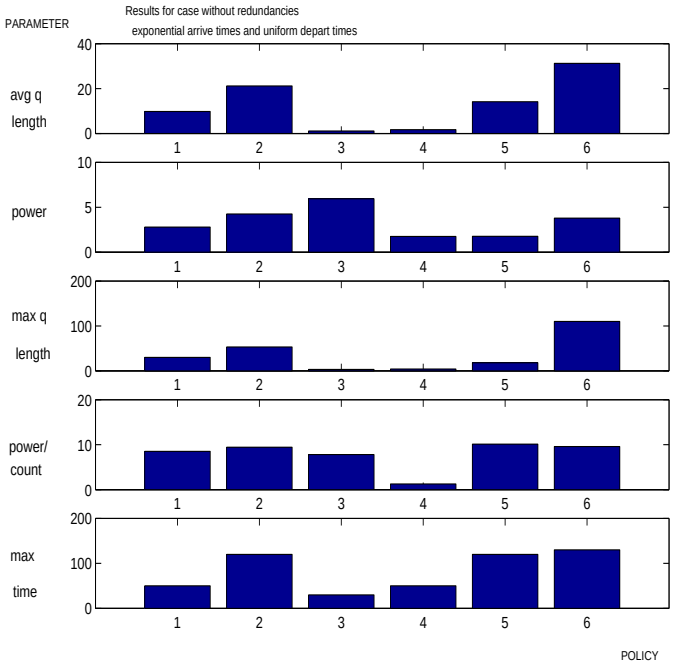

Figure 9.11: **Results for architecture in Figure 4.7 with redundant paths exponential arrival and uniform departure times**

Chapter 10

Appendix 2

Results for experiments with queues to compare server utilization vs loss and queue length characteristics.

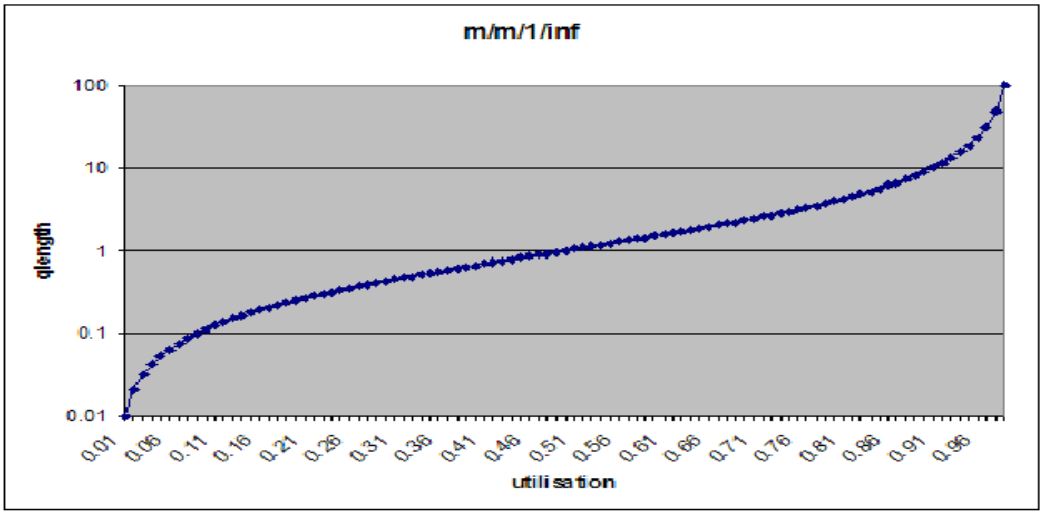

Figure 10.1: **Results for M/M/1 queues**

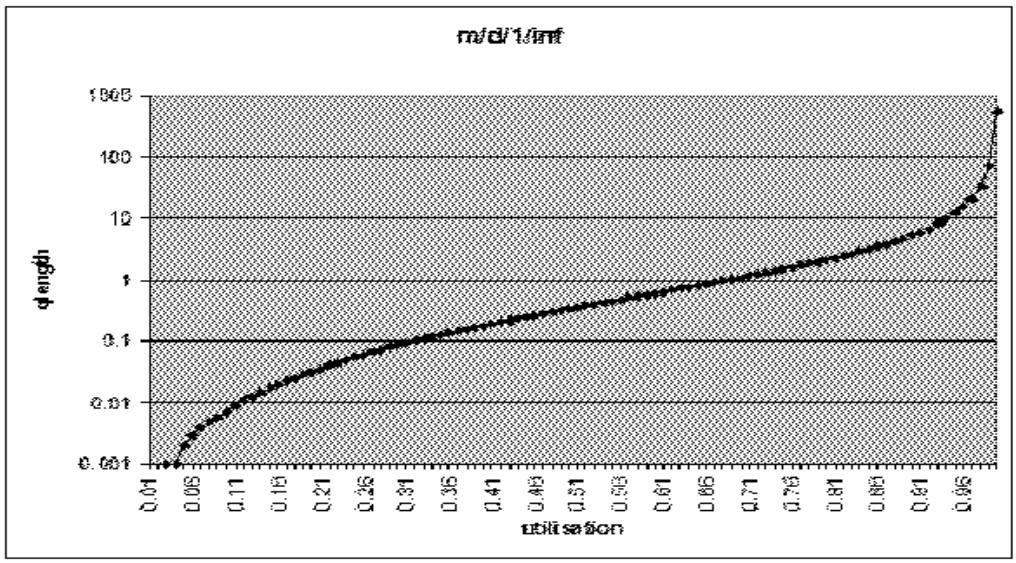

Figure 10.2: **Results for M/D/1 queues**

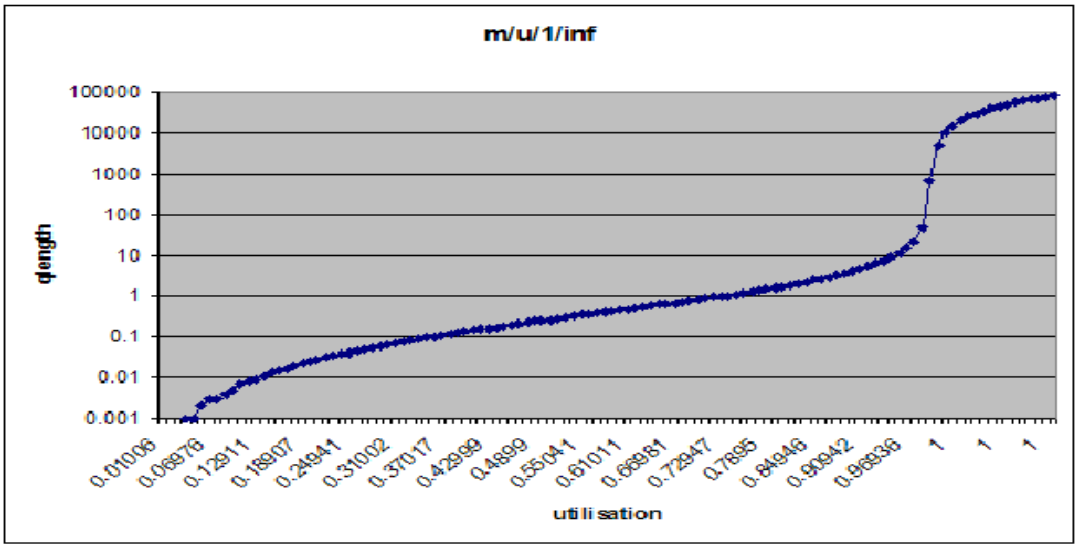

Figure 10.3: **Results for M/U/1 queues**

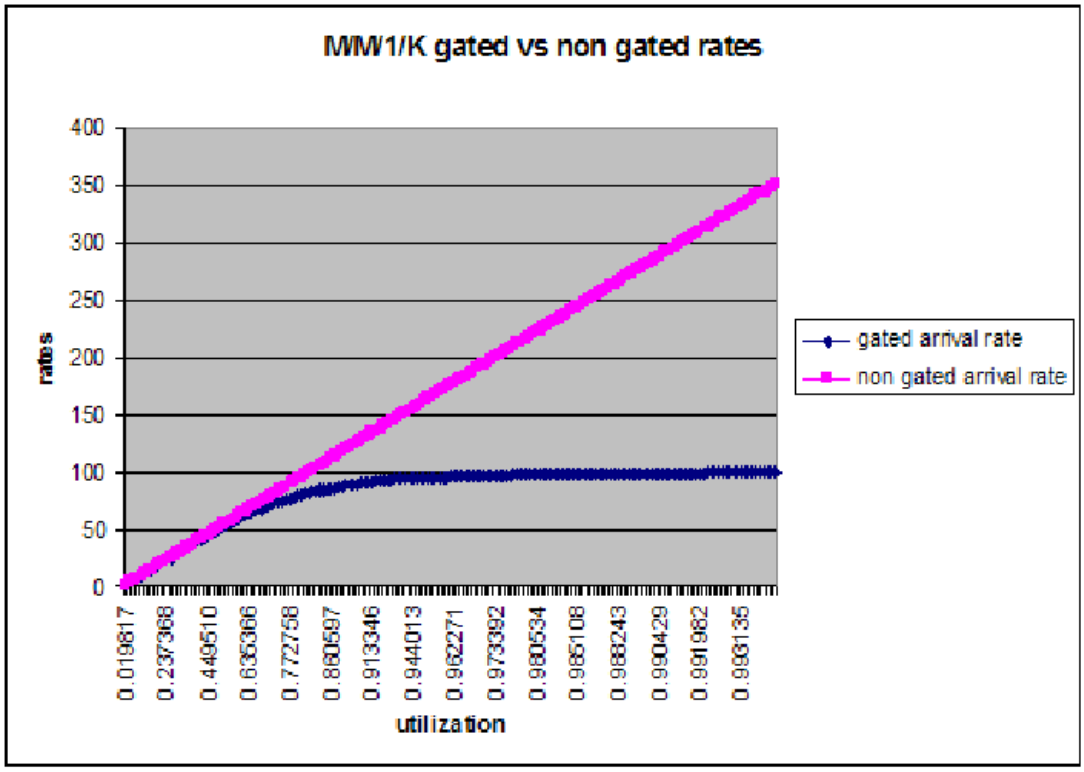

Figure 10.4: **Results for M/M/1/K queues**

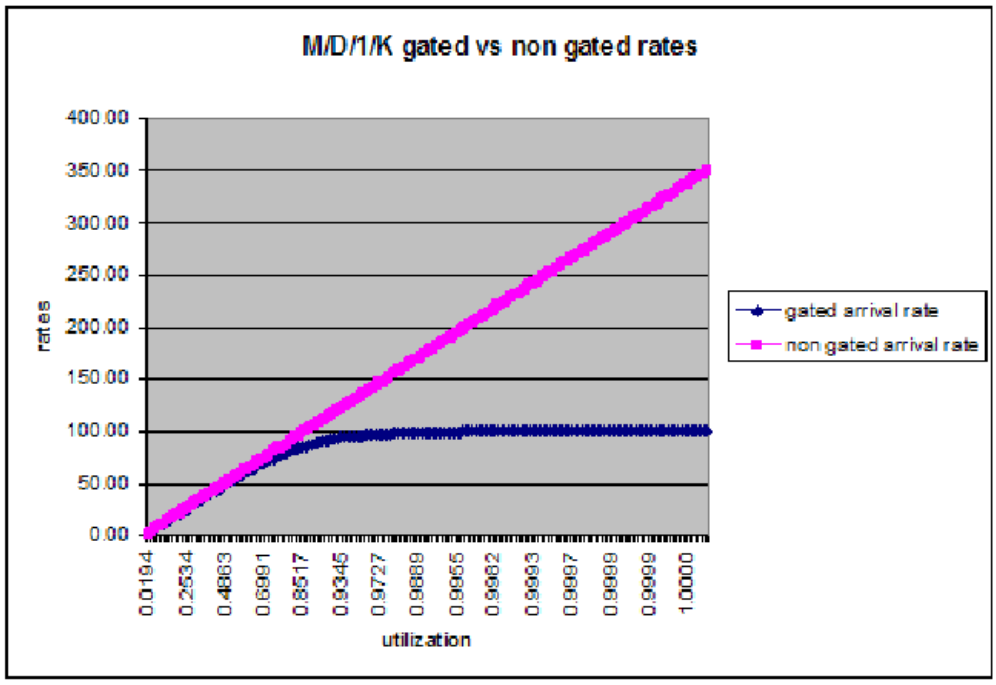

Figure 10.5: **Results for M/D/1/K queues**

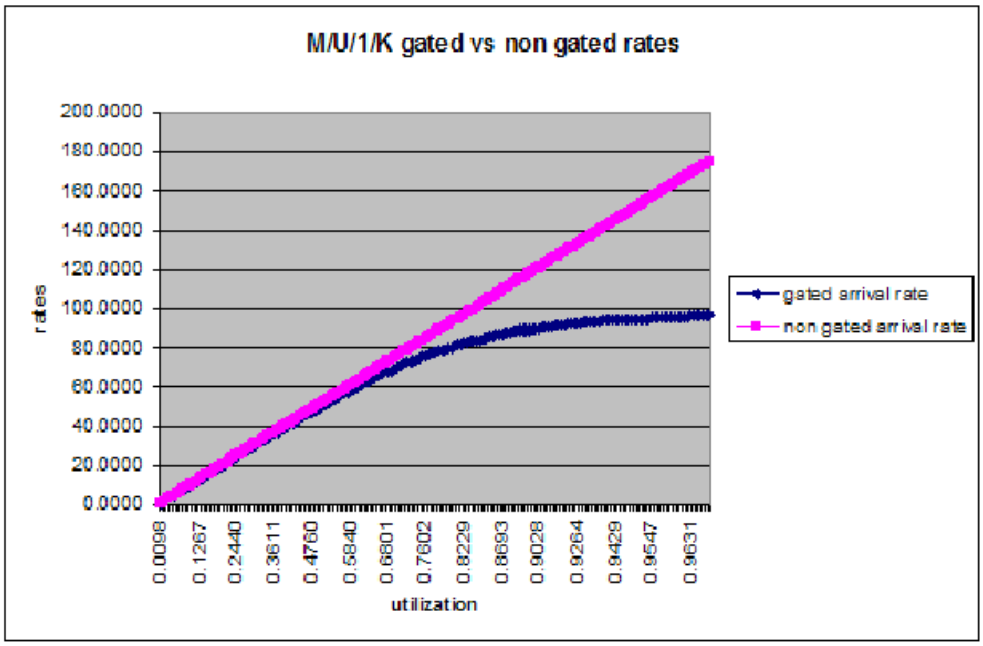

Figure 10.6: **Results for M/U/1/K queues**

Bibliography

[1] S. Agarwala *et al* , " A 600-MHz VLIW DSP", *IEEE Journal of Solid-State Circuits*,Vol. 37,No. 11,pp. 1532-1544, 2002.

[2] A. Alsolaim*et al*, "A Dynamically Reconfigurable System-on-a chip Architecture for Future Mobile Digital Signal Processing", *The European Signal Processing Conference EUSIPCO*, 2000.

[3] A. Andrei, M. Schmitz, P. Eles, Z. Peng, B. Al Hashimi, "Quasi-Static Voltage Scaling for Energy Minimization with Time Constraints", *Proc. Design, Automation and Test in Europe Conference*, 2005.

[4] V. Bala, E. Duerstwald, S. Banerjia, "Dynamo: A Transparent Dynamic Optimization System", *Proc. PLDI*, 2000, pp. 1-12.

[5] L. Benini *et al*, "Policy Optimization for Dynamic Power Management" *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems* ,Vol.18,No. 6, pp. 813-833, 1999.

[6] L. Benini, A.Bogliolo, and G. Micheli. A survey of design techniques for system level dynamic power management. *IEEE Transactions on VLSI Systems*, 8(3):299–316, June 2000.

[7] C. G. Cassandras and S. LaFortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[8] V. Chandra, A. Xu, H. Schmidt, L. Pillegi, "An Interconnect Channel Design Methodology for High Performance Integrated Circuits", *Proc. of Design Automation and Test in Europe*, pp. 1138-1143, 2004.

[9] C.-T. Chen, K. Küçükçakar,"High-levelcheduling Model and Control Synthesis for a Broad Range of Design Applications", *Proceedings of the International Conference on Computer Aided Design*,pp. 236-243,1997

[10] E. Chung *et al*, "Dynamic Power Management for Non Stationary Service Requests", *Proceedings of Design Automation and Test in Europe*, pp. 77-81,1999.

[11] K. Compton *et al*,"Configuration Relocation and Defragmentation for Run-Time Reconfigurable Computing", *IEEE Transactions on on VLSI Systems*,Vol. 10,No. 3,pp. 209-220, 2002.

[12] K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", *ACM Computing Surveys*, Vol. 34, No. 2,pp. 171-210, 2002.

[13] A. Cortes, P. Eles,Z. Peng,"Quasi-Static Scheduling for Real-Time Systems with Hard and Soft Tasks", *Proceedings of Design Automation and Test in Europe*,pp. 1176-1181,2004.

[14] http://www.cray.com/

[15] I. Dalgic, F. A. Toubagi,"Performance Evaluation of Ethernets and ATM Networks Carrying Video Traffic", *Technical Report,Department of Electrical Engineering and Computer Science,Stanford University*, Technical Report No. CSL-TR-96-702,1996.

[16] J.A.Darringer *et al*,"Early analysis tools for System-on-a-chip design", *IBM Journal of Research and Development* ,Vol.46,No.6,pp. 691-707,2002.

[17] T. Dumitras, S. Kerner, R. Marculescu,"Towards On-Chip Fault-Tolerant Communication", *Proceedings of Asia South Pacific Design Automation Conference (ASP/DAC)* ,pp. 225-232,2003.

[18] T. Dumitras, R. Marculescu, "On-Chip Stochastic Communication", *Proceedings of Design Automation and Test in Europe*,pp. 10790-10795,2003.

[19] http://www.es.jamstec.go.jp/

[20] D. Estrin, R. Govindan,J. Heidermann, S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks" , *ACM International Conference on Moblie Computing and Networking*,pp. 263-270,1999.

[21] P. Eles, Z. Peng, K. Kuchinski, and A. Doboli. *System Level Hardware/Software Partitioning using Simulated Annealing and Tabu Search*. Kluwer Academic Publishers, 1997.

[22] X. Fan,C.S. Ellis,A. R. Lebeck,"Memory Controller Policies for DRAM Power Management", *Proceedings of International Conference on Low Power Electronocs and Designs*,pp. 129-134,2001.

[23] E.A.Feinberg, "Constrained Semi-Markov Decision Processes with Average Rewards", *ZOR- Mathematical Methods of Operational Research*, Vol.39,pp. 257-288,1994.

[24] E. Feinberg ," Optimal Control of Average Reward Constrained Continuous Time Finite Markov Decision Processes" *Proceedings of the IEEE Conference on Decision and Control* pp. 3805-3810,2002.

[25] E.A. Feinberg, A. Shwartz,"Handbook of Markov Decision Processes methods and applications", *Kluwer Academic Publishers*,2002.

[26] G. Fettweis *et al*, "Strategies in a cost effective implementation of PDC Half Rate Codec for Wireless Communication", *Proceedings of IEEE Vehicular Technology Conference*, pp. 203-207, 1997.

[27] G. Fettweis, "DSP Cores for Mobile Communication: Where are we going ?", *Proceedings of IEEE International Conference on Acoustics,Speech and Signal Processing* , pp. 279-282,1997.

[28] D. Ganesan *et al*, "An Evaluation of Multi-resolution Storage for Sensor Networks", *Proceedings of the ACM SenSys Conference*, pp. 89-102,2003.

[29] M. Gasteir,M. Glesner, "Bus Based Communication Synthesis on System Level",*Transactions on Design Automation of Electronic Systems* ,pp. 1-11, January,1999.

[30] D. A. Green *et al*,"MDSP: A Modular DSP Architecture for a Real Time 3D laser Range Sensor", *Proceedings of the Electronic Imaging Conference*,2002.

[31] R. Goering,"System-Level Design moves beyond RTL ", *EEtimes*,2000.

[32] L. Hammond *et al* ,"A Single Chip Multiprocessor", *IEEE Computer*,pp.79-85,1997.

[33] S. Hauck, T.W. Fry, J. Kao, "The Chimaera Reconfigurable Functional Unit", *Proc. IEEE Symposium on FPGAs for Custom Computing Machines*, 1997, pp. 87-96.

[34] J. Hesser *et al*,"Three Architectures for Volume Rendering", *Proceedings of Eurographics*,Vol.14,No.3,1995.

[35] J. Henkel, "A Low Power Hardware Software Partitioning Approach for Core-Based Embedded Systems", *Proc. DAC*, 1999, pp. 122-127.

[36] J.L. Hennessey, D.A. Patterson,"Computer Architecture a Quantitative Approach", *Morgan Kaufmann Publications*,1990.

[37] J. Hill *et al*," System Architecture Directions for Networked Sensors" *Proceedings of Iternational Conference on Architectural Support for Programming Languages and Operating Systems*,pp. 93-104, 2000.

[38] IBM, "Blue Logic Cu-08 Voltage Islands", *http://www-03.ibm.com/chips/products/asic/product/v_island.html*, 2005.

[39] IBM, "PowerPC 750FX Microprocessot User's Manual", *http://www-306.ibm.com/chips/techlib/techlib.nsf/products/ Power_PC_750FX_Microprocessor*, 2005.

[40] K. Jerath,"Modelling and Stochastic Analysis of Embedded Systems Emphasizing Coincident Failures,Failure Severity, and Usage Profiles",*Masters Thesis, School of Electrical Engineering and Computer Science, Washington State University*,2002.

[41] E. Jovanov *et al* ,"Real Time Holter Monitoring for Biomedical Signals", *Proceedings of the DSP Education and Technology Conference*,1999.

[42] S. Kallakuri, A. Doboli, and S. Doboli. Stochastic modeling based environment for synthesis and comparison of bus arbitration policies. *Proceedings of IEEE Annual Symposium on VLSI*, pages 199–206, 2004.

[43] J. Kahn, R. Katz,K. Pister, "Next Century Challenges: Mobile Networking for Smart Dust" , *ACM International Conference on Moblie Computing and Networking*,pp. 271-280,1999.

[44] A. Karlin *et al*, "Competitive Randomized Algorithms for Non-Uniform Problems", *Algorithmica*,Vol. 11, No. 6,pp. 542-571,June 1994.

[45] K. Keutzer *et al*, "System Level Design:Orthogonalization of Concerns and Platform-Based Design ", *IEEE Transactions on computer aided design* ,Vol.19,No.12,December 2000.

[46] J. Khan, R. Vemuri, "An Iterative Algorithm for Battery Aware Task Scheduling on Portable Computing Platforms", *Proc. Design Automation and Test in Europe*, 2005, pp. 622-627.

[47] S. Kim *et al*,"Systematic Reliability Analysis of a Class of Application Specific Embedded Software Frameworks ", *IEEE Transactions on Software Engineering*,Vol.30,No.4, pp. 218-230,2004.

[48] L.Kleinrock, "Queueing Systems", *John Wiley and Sons*, 1975.

[49] H. Kloos *et al*, "HIPAR-DSP 16, A Scalable Highly Parallel DSP Core for System on a Chip Video- and Image Processing Applications", *Proceedings of IEEE International Conference on Acoustics,Speech and Signal Processing* , 2002.

[50] A. Krall, "Efficient JavaVM Just-In-Time Compilation", *Proc. ICPACT*, 1998, pp. 54-61.

[51] D. J. Kuck, "Supercomputers and Distributed Computing", *Proceedings of ACM Computer Science Conference*,pp. 34-46,1985.

[52] P. Kumar, M. Srivastav,"Predictive Strategies for Low Power RTOS Scheduling", *Proceedings of the IEEE International conference on Computer Design*,pp. 343-350,2000.

[53] K. Lahiri,A. Raghunathan, S. Dey, "Fast Performance Analysis of Bus Based System-on-Chip Communication Architecture",*Proceedings of the International Conference on Computer Aided Design*, pp. 566-572,1999.

[54] A.M Law,W.D Kelton,"Simulation Modeling and Analysis",*McGraw-Hill*,2000.

[55] H. Lieske *et al*, "Realization of a Programmable Parallel DSP for High Performance Image Processing Applications ", *Proceedings of the Design Automation conference*,pp. 56-61, 2001.

[56] R. Lysecky, F. Vahid, "A Study of Speedups and Comptetiveness of FPGA Soft Processors Cores using Dynamic Hardware/Software Partioning", *Proc. Design Automation and Test in Europe*, 2005, pp. 18-23.

[57] http://www.mathworks.com

[58] A. Maheshwari,W. Burleson, R. Tessier,"Trading Off Transient Fault Tolerance and Power Consumption in Deep Submicron (DSM) VLSI Circuits",*IEEE Transactions on VLSI Systems*, Vol.12, No.3,pp. 299-311,2004.

[59] K. Mai *et al*,"Smart Memories: a modular reconfigurable architecture", *Proceedings of the International Symposium on Computer Architecture*,pp. 161-171,2000.

[60] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson, "Wireless sensor Networks for Habitat Monitoring", *Proc. ACM International Worshop on Wireless Sensor Network Applications*, 2002.

[61] H. McDermott,"A Programmable Sound Processor for Advanced Hearing Aid Research", *IEEE Transactions on Rehabilitation Engineering*,Vol.6,No.1,pp. 53-59,1998.

[62] B. Mei *et al*,"DRESC: A Retargetable Compiler for Coarse - Grained Reconfigurable Architectures", *Proceedings of International Conference on Field Programmable Technology* ,pp.166-173,2002.

[63] P. Mosch *et al*," A 660-uW 500 Mops 1 V DSP for a Hearing Aid Chip Set ", *IEEE Journal of Solid-State Circuits*, Vol.35, No. 11, pp.1705-1712,2000.

[64] B. Miramond and J. Delsome. Design space expolration of dynamically reconfigurable architectures. *Proceedings of Design Automation and Test in Europe*, pages 366–371, 2005.

[65] A. Nandi,R. Marculescu,"System Level Power/Performance Analysis for Embedded System Design ", *Proceedings of ACM/IEEE Design Automation Conference*,pp.599-604 ,2001.

[66] T. Ohya *et al*, "Pitch Synchronous Innovation CELP (PSI-CELP) PDC Half Rate Speech Codec ", *Technical Report IEICE,RCS93-78* , pp. 63-70, 1993.

[67] Y. Okumura *et al*, "A study of DSP circuits Applied to Speech Codec for Digital Mobile Communications *Proceedings of IEICE* , B-294,pp. 2-294, 1993.

[68] G.A.Palegolo, L.Benini, A.Bogliolo, G.De Micheli, "Policy Optimisation for Dynamic Power Management", *Proceedings of the Design Automation Conference*,pp. 182-187,1998.

[69] S. Park *et al* , " Design of Wearable Sensor badge for Smart Kindergarten", *Proceedings of International Conference on Wearable Computers*,2002.

[70] A. Pelkonen,K. Masselos,M. Cupak,"System-Level Modeling of Dynamically Reconfigurable Hardware with SystemC", *Proceedings of the International Parallel and Distributed Processing Symposium* ,pp. 174 ,2003.

[71] H. Pfister, A. Kaufman, F. Wessels,"Towards a Scalable Architecture for Real Time Volume Rendering", *Proceedings of the 1995 Eurographics Workshop on Graphics Hardware*,pp.123-130,1995.

[72] A. Pinar, B. Hendrickson,"Interprocessor Communication with Limited Memory", *IEEE Transactions on Parallel and Distributed Systems*,Vol.15,No.7,pp. 606-616,2004.

[73] M. Potkonjak, I. Hong, M. Srivastava, "On-line Scheduling of Hard-Real Time Tasks on Variable Voltage Processors", *Proc. of International Conference CAD*, 1998, pp. 653-656.

[74] Q. Qiu *et al*, "Dynamic Power Management Based on Continuous Time Markov Decision Processes" *Proceedings of the 36th Design Automation Conference* , pp. 555-561, 1999.

[75] Q. Qiu,Q. Wu,M. Pedram, "Dynamic Power Management of Complex Systems Using Generalised Stochastic Petri Nets" *Proceedings of the 37th Design Automation Conference*, pp. 352-356, 2000.

[76] Q. Qiu, Q.Wu, M.Pedram, "Stochastic Modeling of a Power-Managed System: Construction and Optimisation", *IEEE Transactions on Computer Aided Design*, Vol. 20, No. 9, pp. 1200-1217, 2001.

[77] M. Rahimi, R. Pon, W. Kaiser, G. Sukhatme, D. Estrin, M. Srivastava, "Adaptive Sampling for Environmental Robots", *Proc. International Conference on Robotics and Automation*, 2004.

[78] S. Rixner *et al*,"A Bandwidth Efficient Proccessor for Media Processing", *Proceedings of International Symposium on Microarchitecture* ,pp. 3-13,1998.

[79] T. G. Robertazzi, "Computer Networks and Systems: Queueing Theory and Performance Evaluation", *Springer-Verlag*, 2000.

[80] B. Salefski, Levent Kaglar, "Reconfigurable computing in Wireless", *Proceedings of the 38th Design Automation Conference*,pp. 178-183,2001.

[81] N. Shah, "Understanding Network Processors",*MS Thesis, University of California Berkely*,2001.

[82] T. Sheldon, S. Jerath,"Assessing the Effect of Failure Severity, Coincident failures and Usage-Profiles on the reliability of embedded control systems",*Proceedings of the ACM Symposium on Applied Computing*,pp. 826-833,2004.

[83] G.J.M. Smit, M. Bos, P.J.M. Havinga ,M. Smit, "Reconfigurable Mobile Multimedia Systems" , *Proceedings of the ProRISC workshop on Circuits ,Systems and Signal Processing(ProRISC)*,pp. 431-436,1999.

[84] M. Srivastav, "Sensor Node Platforms and Energy Issues", *ACM International Conference on Mobile Computing and Networking*,Tutorial, 2002.

[85] G. Stitt, F. Vahid, "Hardware/Software Partitioning of Software Binaries", *Proc. ICCAD*, 2002, pp. 164-170.

[86] G. Stitt, R. Lysecky, F. Vahid, "Dynamic Hardware/Software Partitioning: A First Approach", *Proc. DAC*, 2003, pp. 250-255.

[87] M.T.J. Strik,A.H. Timmer,J.L.van Meerbergen,Gert-Jan van Rootselaar,"Heterogenous Multi-Processor for the Management of Real-Time Video and Graphics Streams",*IEEE Journal of Solid State Circuits*,Vol.35,No.11,pp. 1722-1731,2000.

[88] R. Szewyzyck*et al*,"Habitat Monitoring with Sensor Networks", *Communications of the ACM*,pp. 34-40,2004.

[89] MPEG-2 Systems Committee." MPEG-2 systems working draft. ISO/IEC/JTC1/SC29/WG-11-N0501",1993 .

[90] D.Talla ,"Architectural Techniques to Accelarate Multimedia applications on General-Purpose Processors ", *Phd Thesis,ECE Department, UT Austin*,2001.

[91] A. Tanenbaum, "Modern Operating Systems", *Prentice Hall PTR*, 2001.

[92] D. Taubman *et al*, 'Multi-rate 3-d subband coding of video", *IEEE Transactions on Image Processing* , Vol. 3 , No. 5, pp. 572-588,1994.

[93] D. Taubman *et al*, "Orientation adaptive subband image coding", *IEEE Transactions on Image Processing*, Vol. 3, No. 5, 1998.

[94] C. Taylor *et al*, "Run Time Allocation of Buffer Resources to Maximize Video Clip Quality in a Wireless Last-Hop System ", *Proceeding of IEEE International Conference on Communications* , pp.3081-3085,2004.

[95] C. Taylor *et al*, "Orbit: An Adaptive Data Shaping Technique for Robust Wireless Video Clip Communication", *Proc. of Asilomar Conference on Signals,Systems and Computers* , pp. 1567-1571,2003.

[96] http://www.tinyos.net/

[97] http://focus.ti.com/docs/apps/appshomepage.jhtml

[98] TMS320C6201 Fixed Point DSP, Datasheet,Texas Instruments, 2004.

[99] N. Thepayasuwan, V. Damle , A.Doboli " Bus Architecture Synthesis for Hardware Software Codesign of Deep Submicron Systems on Chip",*Proceedings of IEEE International Conference on Computer Design (ICCD)*, pp 126-133, 2003.

[100] N. Thepayasuwan, A. Doboli, " Layout Conscious Bus Architecture Synthesis for Deep Submicron Systems on Chip ",*Proceedings of Design Automation and Test in Europe*,pp 108-113, 2004.

[101] I. Troxel *et al*, "Design and Analysis of a Dynamically Reconfigurable Network Processor", *Proceedings of the IEEE Conference on Local Computer Networks (LCN)*,pp 483-494, 2002.

[102] G. Varatkar, R. Marculescu, "On-Chip Traffic Modeling and Synthesis for MPEG-2 Video Applications", *IEEE Transactions on VLSI Systems* , Vol.12, No.1,pp. 108-119,2004.

[103] I. Verbauwhede *et al*,"Architectures and Design Techniques for Energy Efficient Embedded DSP and Multimedia Processing", *Proceedings of Design Automation and Test in Europe*,pp. 988-993,2004.

[104] S. Vincentelli *et al*, "Platform Based Design", *EEtimes*, 2004.

[105] A. Wang, A. Chandrakasan,"Energy Efficient DSPs for Wireless Sensor Networks", *IEEE Signal Processing Magazine*,Vol. 19, No. 4,pp. 68-78, 2002.

[106] T. Wenisch, R. Wunderlisch, B. Falsafi and J. Hoe *Simulation Sampling with Live-points. Proc. of ISPASS*, pages 2–12, 2006.

[107] Y. Weng, A. Doboli,"Smart Sensor Architecture Customized for Image Processing Applications", *Proceedings of the IEEE Symposium on Real-Time and Embedded Technology and Applications*,pp. 396-403,2004.

[108] Y. Weng *et al*,"Dynamic Architecture Adaptation to Improve Scalability of Sensor Networks: A Case Study for a smart Sensor for Face Recognition", *Selected for publication in IEEE Real-Time Systems Symposium*,2004.

[109] Y.Weng, A. Doboli, "Smart Sensor Architecture Customized for Image Processing Applications", *Proc. IEEE Real-Time and Embedded Technology and Embedded Applications*, 2004, pp. 396-403.

[110] B. C. Williams *et al*,"Model Based Programming of Intelligent Embedded Systems and Robotic Space Explorers ", *Proceedings of the IEEE* ,Vol. 91,No.1,pp. 2003.

[111] B. C. Williams, P. Nayak, "A Reactive Planner for a Model-Based Executice", *Proceedings of the International Joint Conference on Artificial Intelligence* ,1997.

[112] W.Wolf,"An Architectural Co-Synthesis Algorithm for Distributed, Embedded Computing Systems", *IEEE Transcations on VLSI Systems*, pp. 218-229,1997.

[113] M. Writhlin, "Disc: The Dynamic Instruction Set Computer. FPGAs for Fast Board Developement and Reconfigurable Computing", *Proc. of SPIE 2607*, 1995, pages 92-103.

[114] R. Wunderlisch, T. Wenisch, B. Falsafi and J. Hoe SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling. *Proceedings of ISCA*, pages 84–97, 2003.

[115] http://www.xbow.com/

[116] P. Yang, F. Catthoor, "Pareto-Optimization-Based Run-Time Task Scheduling for Embedded Systems", *Proc. of CODES+ISSS*, 2003, pp. 120-125.

[117] T. Yen , W. Wolfe,"Communication Synthesis for distributed embedded Systems",*Proceedings of the International Conference on Computer Aided Design*, pp. 288-294,1995.

[118] J. Yi, D. Lilja, and D. Hawkins. Improving Computer Architecture Simulation Methodology by Adding Statistical Rigor. *IEEE Transactions on Computers*, Vol.54, No.11, pages 1360-1373, 2005.

[119] J. Zhao, R. Govindan, D. Estrin, "Computing Aggregates for Monitoring Wireless Sensor Networks", *Proceedings of IEEE Symposium on Sensor Network Protocols and Applications* ,pp. 139-148, 2003.