**KTH Information and
Communication Technology**

Fuzzy Flow Regulation for Network-on-Chip based Chip Multiprocessors Systems

YUAN YAO

Master's Thesis in System-on-Chip Design, KTH
Supervisor: Zhonghai Lu
Examiner: Zhonghai Lu

TRITA-ICT-EX-2014:171

Acknowledgements

Special thanks to Associate Professor Zhonghai Lu at School of ICT, KTH. Freedom and good advice are essential to success in any academic research. All the other gratitude expressing words are to my parents. For an endless list of reasons. This thesis is dedicated to them with the utmost love and respect.

Abstract

As large uniprocessors are no longer scaling in performance, chip multiprocessors (CMP) become the mainstream to build high-performance computers. CMP chips integrate various components such as processing cores, L1 caches and L2 caches (some also contain L3 caches, for example, in the IBM Power7 multicore processor) together, and multiple CMP chips with external memory banks make up a CMP system. As buses (although long the mainstay of system interconnect) are unable to keep up with increasing performance requirements, network-on-chip (NoC) offers an attractive solution to this communication crisis and is becoming the pervasive interconnection network in CMPs.

In NoC based CMP systems, regulating traffic flows has been shown to be an effective means to improve communication performance and reduce buffer requirements. However, existing flow regulation policies such as the ones describe in [8] and [9] are all static. The parameters (σ , ρ) of the regulators are hard-coded during system configuration, where σ bounds the traffic burst and ρ the traffic rate. Although static flow regulator can be used as a design instrument for System-on-Chip (SoC) architects to control quality-of-service and achieve cost-effective communication, the drawbacks from its static property cancel the gains in some situations.

In this thesis, we design a fuzzy flow regulation mechanism for network-on-chip based CMPs. Being different from static flow regulation policy, our system makes regulation decisions dynamically according to the state of interconnection network. We use fuzzy logic to mimic the behaviors of an expert that validly controls the admission of input flows, with the aim of making better use of on-chip resources and decreasing communication delays.

We implement and test our design under Multi-facet's General Execution-driven Multiprocessor Simulator (GEMS), which creates a platform that is similar to real CMP environment. Hardware imitating models such as L1 caches, L2 caches and memory banks help us to test our design thoroughly and comprehensively.

The experiments are done with both closed-loop and open-loop methods. Comparisons have been made between our design and static regulation policy. The results show that our fuzzy flow regulation system can make good regulation policy with all the testing cases.

Contents

Contents	v
List of Figures	vii
List of Tables	ix
1 Introduction	1
1.1 Flow Regulation in CMP	1
1.2 Static Flow Regulation	4
1.3 Problem Statement	5
1.4 Fuzzy Control Basics	8
1.4.1 Precise Control to Imprecise Control	8
1.4.2 Membership Functions and Rules	10
1.5 Congestion Control Basics	12
1.6 Summary	14
2 Fuzzy-based Regulation Design	15
2.1 Design Structure	15
2.2 Fuzzy Controller Design	16
2.2.1 Fuzzification Unit Design	18
2.2.2 Rule Set Design	19
2.2.3 Inference Unit Design	20
2.2.4 Defuzzification Unit Design	22
2.3 Network State Recognizer Design	23
2.4 Regulator Design	28
2.5 Summary	28
3 Implementation and Experiment Results	29
3.1 System Level Simulation	29
3.2 Implementation Details	30
3.3 Experiment Setup	32
3.3.1 CMP Architecture Parameters	32
3.3.2 Open-loop Measurement	34
3.3.3 Closed-loop Measurement	36

3.4	Experiment Results	37
3.4.1	Results of Open-loop Method with Uniform Random Traffic .	37
3.4.2	Results of Open-loop Method with Bit Permutation Traffic .	44
3.4.3	Discussion of Open-loop Results	49
3.4.4	Results of Closed-loop Method	51
3.5	Summary	53
4	Conclusions and Future Works	55
4.1	Conclusions	55
4.2	Further Works	56
A	Gems Configuration List	59
	Bibliography	61

List of Figures

1.1	A Real CMP Chip (IBM Power7 Processor)	1
1.2	Chip Multiprocessors System	2
1.3	Network Flow Regulator	4
1.4	Leaky Bucket Flow Regulator	5
1.5	Experimental CMP Structure	7
1.6	Link Utilization against Injection Rate	9
1.7	Latency against Injection Rate	10
1.8	Fuzzy Logic Processing Example	11
1.9	Example of Topology and Flows	13
2.1	Flow Regulation with Fuzzy Control	15
2.2	Structure of Fuzzy Controller	17
2.3	Membership Function for Fuzzification Unit	18
2.4	Computation of $\mu(e(t_1))$	22
2.5	Membership Function for Latency	24
2.6	Membership Function for Link Utilization	24
2.7	Membership Function for Composed Premise	26
2.8	Defuzzification Function	26
2.9	Fuzzification and Defuzzification Progress	26
2.10	Defuzzification of Three Samples	27
2.11	Control Surface for Network State Recognizer	27
2.12	Regulator in the Fuzzy Regulation System	28
3.1	System Class Diagram	31
3.2	Physical Architecture of the Experiment CMP	33
3.3	Measurement Setup for Open-loop Measurement	34
3.4	Traffic Patterns for Bit Permutation	35
3.5	Measurement Setup for Closed-loop Measurement	36
3.6	Accumulative Packets Curve	37
3.7	Accumulative Packets for the Four Flows	38
3.8	Injection Rate for the Four Flows	39
3.9	Effects of Fuzzy Controller for Uniform Random Traffic	39
3.10	Delay Histograms for Static Regulated Flows	41

3.11 Delay Histograms for Unregulated Flows	42
3.12 Delay Histograms for Fuzzy Regulated Flows	43
3.13 Link Utilization for Regulated Flows	44
3.14 Effects of Fuzzy Controller for Bit Permutation Traffic	45
3.15 Delay Histograms for Static Regulated Flows	46
3.16 Delay Histograms for Unregulated Flows	47
3.17 Delay Histograms for Fuzzy Regulated Flows	48
3.18 Link Utilization for Regulated Flows	49
3.19 Delay Histogram Comparison	50
3.20 Composition of Delays	50
3.21 Effects of the Regulation Policies on Different Injection Rates	51
3.22 Results of Closed-loop Measurement	52

List of Tables

1.1	Experiment Results for Random Traffic	6
1.2	Experiment Results for Bit Permutation Traffic	8
2.1	Rule Table of the Fuzzy Controller	20
2.2	Rule Table of the Empty Network	21
2.3	Rule Table of the Heavily Saturated Network	21
2.4	Rule Table of the Network State Recognizer	24
3.1	Delays from Different Regulation Policies	40
3.2	Delays from Different Regulation Policies	45
A.1	Parameters for GEMS	59
A.1	Parameters for GEMS	60

Chapter 1

Introduction

1.1 Flow Regulation in CMP

As large uniprocessors are no longer scaling in performance, chip multiprocessors (CMP) become the mainstream to build high-performance microprocessors. CMPs are widely used across many application domains including general-purpose CPU, embedded systems, digital signal processing (DSP), and graphics calculation.

Figure 1.1. A Real CMP Chip (IBM Power7 Processor)

Figure 1.2. Chip Multiprocessors System

A CMP chip contains various components such as processing cores, L1 caches and L2 caches (L3 caches are optional). Multiple CMP chips with some external memory banks make up a multicore system. These chips and memory banks are often connected by an interconnection network, which is usually organized in a regular topology such as torus and mesh. Figure 1.1 shows a real CMP chip (IBM's Power7 processor) and Figure 1.2 shows the structure of a complete multicore system. Notice that the multicore system we use in our thesis does not contain L3 caches. The system consists of N CMP chips and M external memory banks. Communications between different CMP chips and external memories are done by the interconnection network. With limited resources such as bandwidth and buffers, the interconnection network becomes the bottleneck of the system when the communication work load becomes high in scenarios such as radio application or GPU processing. Thus, the focus of this thesis is on how to improve the performance of the interconnection network inside a multicore system, especially when the interconnection network is under heavy work load or near saturated state. We assume that our network is packet switched instead of circuit switched.

The performance of an interconnection network is intimately related to the methods employed to admit traffic into the network. In multicore system, data packets

get “blocked” unless resources are available. Packets can receive very bad service when the network is heavily loaded. Thus in building a network, it is crucial to examine the problem of how to regulate the incoming traffic. Usually, this network injection control function is called flow regulation [8, 4]. **The goal of this thesis is to make good flow regulation systems, depending on the level of congestion in the network, through a fuzzy system.**

Flow regulation is particularly important in multicore systems for two reasons: First, more is demanded of the network in terms of performance guarantees, and second, many more packets are submitted to the network, leading to potentially worse problems of buffer overflow if there is no flow regulation mechanism. Unfortunately, there is no formal definition or expression to calculate when a network is saturated. Most of the time, network saturation is recognized through the measurement of other metrics such as packet latency or link utilization. For example, when packet latency is high the network is saturated, and when packet latency is low or near the zero-load latency the network is empty. But how about a network with packet latency neither low nor high? The fuzzy property of the measurement of network saturation manifests that it should be handled by fuzzy logic.

One property of flow regulation in CMP is that it also has the problem of control latency, like any other control system. Control latency is the time between the generation of a control signal and the control signal begins to take effect. For example, in our fuzzy control system, the flow regulation mechanism can react when the network is saturated. The fuzzy controller changes the regulation policy and tries to recover the network from saturation. This progress needs some time to be done and this time is called the controller latency. Many factors can effect the control latency, such as the controller overhead, response time of each component in the control system, the system clock frequency, delays of the routers, etc. In order to tolerate the control latency, we design our fuzzy flow regulation mechanism using sampling windows. The fuzzy controller does not make new flow regulation policy in every cycle. It samples different input signals during a sampling window, and makes new policy when the sampling window is over. The sampling window is the time for the control signal to take effect. More about the design of our system is in Chapter 3.

Another property of flow regulation in CMP is that it can tolerate imprecise control. Unlike precise control, imprecise control does not require the control signal to be precise. This property is due to the fact that the effects of flow regulation are manifested by statistics metrics such as average packet delay and link utilization. We do not need to control the delay of each packet to achieve an exact value.

The remaining of this thesis is organized in the following way. Chapter 1 introduces the problem we are going to solve as well as some basic knowledge about fuzzy logic and congestion control theory. Such knowledge is essential of our fuzzy flow regulation system. Chapter 2 details the design of each component inside the system. Chapter 3 describes the implementation and the experiment results. Chapter 4 concludes this thesis, and posts potential expansions that can be achieved in the future work.

Figure 1.3. Network Flow Regulator

1.2 Static Flow Regulation

In multicore systems, a packet flow is a sequence of packets from a request component to a response component traveling through the interconnection network. Flow regulation is a mechanism for traffic management. It is a process of managing the peak rate (not average rate) as well as the burst of data transmission between two nodes to provide an acceptable network performance. For example, a flow regulator in the sender side can prevent a fast sender from outrunning a slow receiver. With a well-designed flow regulation policy, communication networks can avoid congestion and provide good service.

A network flow regulator reshapes the input flow to achieve a desired output flow. There are many references dealing with the research of network flow regulator, such as [8] and [9]. The functionality of a flow regulator can be abstracted in Figure 1.3. Flow $A(t)$ is the original flow, and flow $D(t)$ is the regulated flow. In multicore systems, each core initiates a memory transaction flow when cache miss happens. On receiving the request, the responsible components (L2 cache for L1 cache miss request, external memory for L2 cache miss request) initiate the response flow. The response flow usually consists of bytes of data, which are injected into the network in a bursty fashion.

Suppose the cache miss rate is ρ' for each core. Due to the uncertainty of program execution and data sharing situations between cores and caches, in some periods ρ' may be too high and the network gets saturated easily. For example when a loop is executed for the first time, a lot of cold misses happen and cores fetch data from the main memory frequently. If there is no flow regulator between cores and the network, the network can quickly get saturated.

Formally, a network flow regulator is a regulator with envelope E , if for any input arrival process A , the departure process D has envelope E (see Figure 1.3). Thus, for all t and τ , $0 \leq \tau \leq t$, $D(t) - D(\tau) \leq E(t - \tau)$, or

$$D \leq D \otimes E \quad (1.1)$$

where $\otimes$ means min-plus convolution. As to more about min-plus algebra, please refer to [8].

Inequality (1.1) is the definition of envelope function $E(t)$, which is a nondecreasing and nonnegative function. Thus, flow regulator is nothing but the implementation of an envelope function. Envelope function decides over any interval of time of length μ , the amount of data brought by the arrival process $A(t)$ is bounded by $E(\mu)$.

Figure 1.4. Leaky Bucket Flow Regulator

Figure 1.4 shows a *buffered leaky bucket* flow regulator [8]. It consists of a token bucket that can hold up to σ tokens, and tokens are fed at rate ρ . Only if there are tokens available, packet is allowed to depart from the regulator, otherwise the packet is buffered in the source buffer. Notice that the output process D satisfies (for $0 \leq \tau \leq t$)

$$D(t) - D(\tau) \leq \sigma + \rho(t - \tau),$$

which is because the maximum amount of packets that can depart is the number of tokens that arrive in $[\tau, t]$ plus the τ .

After the regulation, the injection rate is forced to ρ . In CMP environment, σ is equal to the amount of packets in a data block, since block is the basic unit of data transactions. For example, suppose one data block consists of 6 packets (1 packet for flow control and 5 for data transfer). When a L1 (or L2) cache data request packet is injected into the network, 6 tokens are consumed because the request generates 6 packets. However, when a L1 cache coherence control packet is injected into the network, 1 token is consumed because only 1 control packet is generated. The delayed packets that are waiting for tokens are stored in the source.

Most of existing flow regulation policies are static, such as the ones described in [1] and [2]. Static flow regulation policies are offline configured. The parameters (σ, ρ) of the regulators are hard-coded during system configuration, where σ bounds the traffic burst and ρ the traffic rate. Static flow regulator can be used as a design instrument for System-on-Chip (SoC) architects to control quality-of-service and achieve cost-effective communication.

1.3 Problem Statement

One problem with the static regulation policy is rigid regulation, which has been explained in [3]. In large scale SoCs, complex system communicating reshapes data flow patterns. Static regulator lacks the flexibility to handle flows with varying characteristics. It can also not change the regulation policy according to network

states. This drawback cancels the gains of static regulation in some situations. For example, assuming a static flow regulator is attached to a core with 0.05 packet/cycle as the long term average injection rate. The regulation parameter ρ of the regulator is pre-loaded and remains unchanged. Suppose at time t the communication network becomes saturated. With static flow regulator, the data source keeps injecting packets at 0.05 packet/cycle, which will further worsen congestion in the network.

To give a concrete example of the rigid regulation problem, we performed the following experiments. The experiment has been done with Multi-facet’s General Execution-driven Multiprocessor Simulator (GEMS) Toolset, which was developed by University of Wisconsin-Madison. For a detailed description of GEMS, please refer to [5].

In the experiment, four CMP chips were connected with each other. Each CMP contained four processors/cores, four private L1 caches, four shared L2 cache banks, and one memory bank. The memory bank was controlled by a detailed DDR3 memory controller module and the caches were controlled by a CMP directory coherence protocol. The coherence protocol was supported both by the CMP cache controller and the interconnection network coherence manager. The cache controller and the network coherence manager ensures that changes in the values of shared memories are propagated throughout the system. All these components were connected by a mesh interconnection network. The picture of the experiment platform is in Figure 1.5. For a detailed system configuration parameters, please see “Gems Configuration List” of Appendix A.

The traffic patterns were uniform random traffic and permutation traffic, where uniform random traffic was to simulate load balanced situation and bit permutation was to simulate load unbalanced situation [6, p.49-51]. The routing algorithm was X-Y dimension routing. Each data source injects synthetic packet flows into the network with an average injection rate ρ , which mimics the long term cache missing rate of the processor. For data packets, the packet size is 5 flits (1 head flit with 3 body flits and 1 tail flit). For control packet, the packet size is 1 flit (only 1 head flit). Ratio between data packets and control packets is set to 1/8. Each processor is integrated with a static flow regulator to do flow regulation.

Simulation time was set to long enough (10000000 cycles in our experiment) for network saturation to become obvious (by showing a high network delay value). The experiment results are in Table 1.1 and 1.2.

Table 1.1. Experiment Results for Random Traffic

Average Delay (cycles)	Packet Injection Rate (packet/cycle)					
	0	0.01	0.02	0.03	0.04	0.05
Queuing Delay	47.98	47.98	48.77	194.84	256.04	266.95
Network Delay	1.00	1.00	1.00	9482.49	29660.52	49681.93
Total Delay	48.98	48.98	49.77	9677.32	29916.56	49948.88
Link Utilization	0.0000	0.0897	0.1794	0.1998	0.2003	0.2001
Simulation Time	10000000 Cycles

Figure 1.5. Experimental CMP Structure

Table 1.2. Experiment Results for Bit Permutation Traffic

Average Delay (cycles)	Packet Injection Rate (packet/cycle)					
	0	0.01	0.02	0.03	0.04	0.05
Queuing Delay	47.98	48.17	48.15	147.71	176.83	195.90
Network Delay	1.00	1.00	1.00	10438.10	23094.00	35596.95
Total Delay	48.98	49.17	49.15	10585.81	23270.83	35792.85
Link Utilization	0.0000	0.0883	0.1765	0.1908	0.1896	0.1940
Simulation Time	10000000 Cycles					

In both experiments the network is stable when the injection rate is equal or less than 0.02 packet/cycle. Thus, the maximum acceptable injection rate is:

$$\rho = 0.02 \times (s_i * 0.125 + s_j * 0.875 = 5 * 0.125 + 0.875) = 0.03 \text{ flit/cycle} \quad (1.2)$$

where s_i is the size of data packet, s_j is the size of control packet.

The results show that with static flow regulation, both of the traffic patterns saturate the network. Due to the flow regulator's unawareness of the network state, data sources keep injecting packets when the network is saturated. **Thus, static flow regulator can not provide efficient flow regulation mechanism when the network is beyond saturated point.**

1.4 Fuzzy Control Basics

1.4.1 Precise Control to Imprecise Control

The reason behind the above problem is that the flow regulator lacks saturation or congestion recognition mechanism.

Network saturation has been studied and explained well in [6, p.452-456]. Due to the complexity of NoC, a general formal analysis of network saturation is hard. How a particular network gets saturated depends on various factors such as network topology, routing algorithm, flow control method, and traffic pattern. Manufacturing constraints such as packing also affects whether a network is easy to be saturated or not.

To make the problem simpler, in the perspective of performance analysis, network saturation is captured by the measurement of link utilization and latency (average packet delay).

When the network is not saturated, link utilization goes high as the injection rate increases. When the injection rate is increased beyond saturation, the network is not able to deliver packets as fast as they are being created. Thus, for a stable network, link utilization remains the same beyond saturation. For an unstable network, link utilization drops. The network in Section 1.2 is a stable network, and the link utilization against injection rate is shown in Figure 1.6.

As to latency, latency vs. injection rate starts at the horizontal asymptote of zero-load latency and slopes upward to the vertical asymptote of saturation la-

Figure 1.6. Link Utilization against Injection Rate

tency. At a low injection rate, latency stays at zero-load latency. As injection rate goes high, increasing packets must wait for sources to travel the network, with consequence that latency curve approaches a vertical asymptote. Latency against injection rate is shown in Figure 1.7.

While it is easy for human-beings to recognize network saturation in graphs such as Figure 1.6 and Figure 1.7, it is hard for computers to define precisely when the network is saturated. For example, if we define a network delay of 50 cycles as the threshold for saturation, can we say the network is not saturated when the delay is 49 cycles or 48 cycles? As to link utilization, if the reaching of 0.02 packet/cycle signals a saturation, how about the link utilization reaches only 0.0199 packet/cycle? If there is no such a definition of network saturation, how does a component in the network (such as a flow regulator) make efficient regulation policy based on network state?

In contrast to precise control, fuzzy logic and fuzzy control can be used when truth values are more than one. Fuzzy logic, or fuzzy control, adopts the concept of “Degree of Truth”, which is considered unambiguously true or false. For example, in discrete mathematics, if a is true and b is true, then $a \wedge b$ is true; if a is false and b is false, then $a \vee b$ is false. However, in reality, some proposition might be more or less true, rather than simply true or simply false. For example, consider the proposition “I’m full”.

The philosophy behind this thesis is using the property of fuzzy logic to “more wisely” recognize the network state. When the flow regulator regulates flows, the network state information acts as the reference to the flow regulation policy. Closing the loop of the state recognition and flow regulation, we get a whole fuzzy controller system to control the quality of service of NoC.

Figure 1.7. Latency against Injection Rate

1.4.2 Membership Functions and Rules

The fuzzy controllers are very simple conceptually. They consist of an input stage, a processing stage, and an output stage. The input stage maps inputs through membership functions to “degree of truth” values (which are between 0 and 1). The processing stage invokes each appropriate rule and generates a truth value for each rule, then combines the truth values of different rules. Finally, the output stage converts the combined result back into a specific control output value. Notice here that “degree of truth” and truth values are conceptually different. “Degree of truth” is the result of membership functions, which denotes the “degree” of one input. The truth value is the result of fuzzy rule set, which reflects the confidence of truth of that rule.

A basic application of using fuzzy logic to recognize network saturation is latency measurement. In the input stage, a latency measurement might have several separate membership functions defining particular delay ranges needed to control the flow regulator properly. For example, three membership (as shown in Figure 1.8) functions can be involved in the input stage. The most common shape of membership functions is triangular, although other shapes are also used, but the shape is generally less important than the number of curves and their placement.

In Figure 1.8, three delays (x , y , and z) from three network packets are sensed and input to the fuzzy controller. Their values decide which membership function is involved to handle them. For instance, x and y are handled by membership function *High* and *Middle*, while z is handled both by *Middle* and *Low*. The results of the input stage are “degree of truth” ($de()$ in the figure) of the input values. This progress is also called the fuzzification of the input values.

In the processing stage, appropriate rules are invoked to handle the “degree of truth” of each input. The results are truth values of the network state based on

Figure 1.8. Fuzzy Logic Processing Example

each rule. In this example, we use three rules:

rule 1 : IF (network latency is “High”) THEN (network is “Highly saturated”)

rule 2 : IF (network latency is “Middle”) THEN (network is “Normal”)

rule 3 : IF (network latency is “Low”) THEN (network is “Empty”)

As showed in Figure 1.8, latency x is “High” with “degree of truth” as $de(x)$. This means we believe x is a high latency, with the degree of $de(x)$. Since x is “high”, Rule 1 will be invoked, which concludes that the network is heavily saturated. The truth value of Rule 1 means that we believe Rule 1 is true, with the confidence of $mu(x)$. Notice that “ mu ” is standard fuzzy-logic nomenclature for “truth value”, as it is used in [10]. Similarly, latency y is mapping to $de(y)$, which is interpreted that the network is working normally, with the truth value $mu(y)$.

Latency z is mapping to $mu'(z)$ and $mu''(z)$, according to proper rules. Since

$de'(z)$ is bigger than $de''(z)$, it is more “true” that the network is empty than normal. Making decision based on comparison of truth value is one of the core differences between fuzzy controllers and precise controllers such as proportional-integral-derivative (PID). Based on this principle, z will invoke rule 3 and generate $mu'(z)$ as the truth value.

The last task in the processing stage is combining the truth value generated by the invoked rules. This process is called “defuzzification”. Of all the defuzzification methods, the “centroid” method is very popular, in which the “center of mass” of the result provides the output value. Another approach is the “height” method, which takes the value of the biggest contributor. The centroid method favors the rule with the output of greatest area, while the height method obviously favors the rule with the greatest output value. For other ways of defuzzification, please refer to [10].

In this thesis, the “centroid” method has been used, which can be formally expressed as formula 1.3 and illustrated in Figure 1.8.

$$n = \frac{\sum_i mu(x_i) \times cen(x_i)}{\sum_i mu(x_i)}, \quad (1.3)$$

where $cen(x_i)$ is the center of rules in Figure 1.8. The truth values are the input to defuzzification progress. The “centroid” method then generates the output value, which is used to reflect the network state.

In the output stage, the output value is input to the flow regulator as a reference of the network state. Based on the reference value and the injection rate of local data source, flow regulator generates new flow regulation policies.

The above example only uses network latency as the input value. In our design, we take link utilization as another input. The rule set should also be enriched if a new input value is added. A richer rule set can help make more precise decisions, thus make the flow regulator more efficient. For instance, the following rule is added:

rule 4 : IF (network latency is “Low”) AND (throughput is “High”) THEN (network is “Highly saturated”).

If the rule specifies an AND relationship between the mappings of the two input variables, as the example above does, the minimum of the two is used as the combined truth value; if an OR is specified, the maximum is used.

The hard part in designing an effective fuzzy controller is to figure out what rules actually work correctly in practice. Good rules should help maximizing the link utilization of the network while minimizing the network delay.

1.5 Congestion Control Basics

In this thesis, we use congestion control theory to calculate the reference input of our fuzzy flow regulation method. The modern theory of congestion control was pioneered by Frank Kelly [11], who applied microeconomic theory and convex optimization theory to describe how individuals controlling their own rates can interact

Figure 1.9. Example of Topology and Flows

to achieve an “optimal” network-wide rate allocation. A thorough implementation of the congestion control mechanism is beyond the topic of this thesis. For detailed information, please refer to [7].

The mathematical expression for optimal rate allocation can be described as follows. Let x_i be the rate of flow i , C_l be the capacity of link l , and r_{li} be 1 if flow i uses link l and 0 otherwise. To achieve a maximum global throughput without congesting the network, the optimal rate allocation then satisfies:

$$\text{objective function} : \max_x \sum_i x_i, \text{ such that } R \times X \leq c, \quad (1.4)$$

where R , X and c be the corresponding vectors and matrix.

For example, consider the flows and topology in Figure 1.9. Five nodes are connected by four links (dashed lines a, b, c, and d) in star topology, with three flows (flow 1', flow 2', and flow 3') traveling through the network. Assume the capacity of link is 1 packet/cycle universally. What is the allocation of injection rate that can achieve globally maximum throughput?

Inequality 1.4 can be used to solve this problem. From Figure 1.9, we can see flow 1' only uses link c and link b. Similarly, flow 2' uses link d and link b, with flow 3' uses link a and link b. Thus, equation 1.4 is:

$$\text{objective function} : \max_x x_1 + x_2 + x_3, \text{ such that } R \times X \leq c, \quad (1.5)$$

where R is the matrix denotes the path of each flow, $X = (x_1, x_2, x_3)^T$ and $c = (1, 1, 1, 1)$.

Formula 1.5 can be solved using Lagrange multipliers or Linear programming method. The result shows that when $x_1 = x_2 = x_3 = \frac{1}{3}$ packet/cycle, global maximum throughput is achieved, which is $x_1 + x_2 + x_3 = 1$ packet/cycle. If either x_1, x_2 or x_3 exceeds $\frac{1}{3}$, maximum throughput can also be achieved, but congestion will occur at node 5.

Formula 1.4 is a basic model of congestion control theory. It gives out the optimal allocation policy assuming the network is never saturated. In practical use,

many other factors (such as waiting delay at routers) can affect the allocation of injection rate. Taking these factors adds additional constraints to the solution space, which can further reduce the result of objective function.

In this thesis, the results of Formula 1.4 is used as a reference input for the fuzzy flow regulator. Since these results are calculated using global objective function with assumptions such as zero routing delay, they are the upper bound of injection rate. Data source should not exceed these injection rates. Otherwise, congestion will happen.

1.6 Summary

In this chapter, we explained what is CMP and why it is necessary to have flow regulation mechanism in CMP. We also explained how static flow regulation works and what problems it arises. In order to solve these problems, we introduced our solution: making flow regulation policies based on the state of the network. In order to correctly recognize the state of the network, we designed our flow regulation system based on fuzzy logic. We also introduced some basic knowledge about fuzzy logic and congestion control. This knowledge is essential to understand the design procedures in Chapter 2.

Chapter 2

Fuzzy-based Regulation Design

In this chapter, we will detail the implementation of each component in the fuzzy regulation system.

2.1 Design Structure

Figure 2.1. Flow Regulation with Fuzzy Control

Figure 2.1 shows how flow regulator regulates the injection rate of data source based on fuzzy control logic. Three components (Regulator, Fuzzy Controller, and Network State Recognizer, as surrounded by the dashed line box) work together to achieve the goal. The control signals are labeled in *italic style*. We list the job of each component in the following.

- The network state recognizer is a fuzzy-logic based network state recognizing component. It monitors packet latency and link utilization, and generates network state information *Network State*, which is sent to the fuzzy controller.
- The fuzzy controller is the core of our design. It has three input signals: injection rate of the data source (ρ'), the reference injection rate (*Reference Input*), and the network state information (*Network State*). Its task is to adjust the regulator's regulation policy using fuzzy logic. The new policy is carried by

Adjust Signal to the flow regulator. The reference input is calculated offline by congestion control theory in Chapter 1.

- The regulator is leaky-bucket based, which has been introduced in Chapter 1. The regulator forces the output to keep an injection rate from ρ' to ρ .

In this chapter, we will describe a detailed design of all the above components. This thesis uses the topology in Figure 1.5. For other topologies, the structure of the design remains the same. But parameters such as reference input, fuzzification membership values should be calculated according to the new environment.

2.2 Fuzzy Controller Design

The reference input should be calculated before the design of the fuzzy controller. Without the reference input it is impossible to tune the membership functions of fuzzification unit or design defuzzification methods.

As stated in Chapter 1, congestion control theory is used to calculate the reference input. It is done offline using a pure mathematical method. Values such as link capacity C_l and link usage indicator r_{li} depends on the topology and routing algorithm of the experiment setup.

In Figure 1.5, we calculate the link capacity using GEMS link delay parameters. For connections between data sources (processors, L1 caches, L2 caches) and networks, link capacity is: $\frac{1}{int_node_link_latency}$, which is 1 flit/cycle (assuming $int_node_link_latency$ is 1 clock cycle). For connections between external memory banks and network, link capacity is: $\frac{1}{ext_node_link_latency}$, which is 0.05 flit/cycle (assuming $ext_node_link_latency$ is 20 clock cycles). For network links (links between network routers), the link capacity is also 1 flit/cycle. Thus, links between external memory and the interconnection network are the bottleneck links. The parameters can be found in Appendix A.

We further assume that every processor (with L1 cache inside) can possibly communicate with every L2 cache bank. This is because the cache miss of one processor may get satisfied with any L2 cache bank(s). Similarly, we assume every L2 cache bank can communicate with any memory bank. Based on this assumption, there are totally $16 \times 16 + 16 \times 4 = 320$ flows in matrix X of inequality 1.4.

Since the routing algorithm is deterministic routing (X-Y dimension), the path of each flow can also be calculated offline, and gives us matrix c of inequality 1.4.

We use Matlab to solve inequality 1.4. The results show that the optimal injection rate for every processor is 0.025 packet/cycle, with global throughput $0.025 \times 16 = 0.4$ packet/cycle.

The block diagram of the fuzzy controller is shown in Figure 2.2. The fuzzy controller is composed of the following four elements:

1. A rule-base (a set of IF-THEN rules), which contains a fuzzy logic quantification about how to achieve good flow control.

Figure 2.2. Structure of Fuzzy Controller

2. An inference mechanism (also called an “inference engine” or “fuzzy inference” module), which makes decision in interpreting and applying rules about how best to control the injection rate.
3. A fuzzification interface, which converts controller inputs into information that the inference mechanism can easily use to activate and apply rules.
4. A defuzzification interface, which converts the conclusions of the inference mechanism into actual inputs for the process.

Notice that *network state* is input to the fuzzy controller as another reference rule, which bypasses the fuzzification progress. The *Network info.* includes information such as average packet delay per sampling window and average link utilization per sampling window. Such information is obtained from the statistics unit of GEMS, where the statistics about the network is collected and the relevant value calculated.

The grey colored components are associated with data source P_0 . Each data source in the CMP owns one fuzzy controller and one flow regulator, while the network state recognizer is shared by all the data sources within the same CMP.

The injection rate ρ' of L1 and L2 caches is the same as the caches' miss rate. For example, when one processor encounters a L1 cache miss (L1 cache is part of the processor in modern computer architecture), the cache controller issues a memory access transaction, which travels through the interconnection network to the desired L2 cache controller. If the cache miss satisfies with a local shared L2

Figure 2.3. Membership Function for Fuzzification Unit

cache, then only intra-chip network is used. Otherwise, if the cache miss satisfies with a remote shared L2 cache, then on-chip network is also used. The cache miss/hit ratio depends on the coherence protocol being used. In this thesis, the cache coherence protocol is directory based MOESI protocol. For more information about MOESI protocol, please refer to [12].

2.2.1 Fuzzification Unit Design

For the fuzzification unit, the difference ($e(t)$) between the reference injection rate and the injection rate of data source (ρ') is calculated first. $e(t)$ then is input to the fuzzification unit which further communicates with the inference mechanism. $e(t)$ is calculated using the following formula:

$$e(t) = \rho'(t) - \rho_r, \quad (2.1)$$

where $\rho'(t)$ is the injection rate of the data source at time t , ρ_r is the reference injection rate.

For example, assume that the reference input is 0.02 packet/cycle and ρ' is 0.025 packet/cycle. Assuming further that the network state is empty, then the interference unit tries to relax the regulation policy and allows more packets to enter the network. Otherwise, if the network state is highly saturated, the interference unit will tighten the regulation policy to decrease the pressure of the network.

The fuzzification of the inputs is handled by membership functions. Tuning of the membership function affects the accuracy of fuzzification. What values to use and how many membership functions the fuzzification unit has depends on cases. For advice about how to make efficient tuning, please refer to [10, p. 101 - 104].

In section 2.2, we have calculated that the optimal injection rate for each processor is 0.025 packet/cycle. With seven membership functions, the fuzzification unit has the function showed in Figure 2.3.

From the figure we can see that $e(t)$ spans from $-\infty$ to ∞ , which reflects that ρ' maybe too high or too low from the reference value.

Assume at some time point T , $e(T)$ is 0.003, as shown in Figure 2.3. The membership function will generate two “Degree of truth” values, one is $de(e(T)) = \alpha$ and the other is $de(e(T)) = \beta$. Since α is bigger than β , the fuzzification unit

interprets that the current injection rate ρ' is slightly high (with “Degree of truth” equals to α) than the reference value.

If $e(T)$ is at the cross point of two membership functions, it can be interpreted in either way. For example, when $e(T) = -0.0075$, ρ' is either *Low* or *Slightly low* than the reference input.

2.2.2 Rule Set Design

Rule set design is to use linguistic quantification to specify a set of rules (a rule-base) that captures the expert’s knowledge about how to control the plant. In particular, for the fuzzy controller, we design the following rules:

1. IF error is *very low* THEN adjust signal is *very loose*. This rule means if ρ' is very low, then the fuzzy controller casts no flow regulation policy. If there is no congestion in the network, the processor can pass all the request packets to the network immediately.
2. IF error is *low* THEN adjust signal is *loose*. This rule means if ρ' is low, then the fuzzy controller casts a loose flow regulation policy. The regulator has no effects on the original ρ' . But if ρ' is increasing during the next control window, the flow regulator limits the increase.
3. IF error is *slightly Low* THEN adjust signal is *slightly loose*. This rule means if ρ' is slightly low, then the fuzzy controller casts a slightly loose flow regulation policy. The regulator has no effects on the original ρ' , but the limitation on the increase is further strengthened.
4. IF error is *equal* THEN adjust signal is *zero*. This rule means if ρ' is equal to the reference injection rate, then the fuzzy controller casts a zero-increase flow regulation policy. The regulator has no effects on the original ρ' , but forbids any increase of ρ' .
5. IF error is *slightly high* THEN adjust signal is *slightly tight*. This rule means if ρ' is slightly higher than the optimal injection rate, then the fuzzy controller casts a slightly tight flow regulation policy. The regulator decreases ρ' over the next control window, and forbids any increase of ρ' .
6. IF error is *high* THEN adjust signal is *tight*. This rule means if ρ' is higher than the optimal injection rate, then the fuzzy controller casts a tight flow regulation policy, which forbids all increase and further decreases the original ρ' .
7. IF error is *very high* THEN adjust signal is *very tight*. This rule means if ρ' is much higher than the optimal injection rate, then the fuzzy controller casts a very tight flow regulation policy. The flow regulator shuts down the corresponding data source, which means forcing ρ to be 0.

Each of the seven rules listed above is a “linguistic rule” because it is formed solely from linguistic variables and values. Since linguistic values are not precise representations of the underlying quantities that they describe, linguistic rules are not precise either. They are simply abstract ideas about how to achieve good control that could mean different things to different people. In order to use them practically, the qualification in Rule Table 2.1 should be used.

Table 2.1. Rule Table of the Fuzzy Controller

			<i>Regulation Policy</i>
		ρ	ρ'
$e(t)$	-0.015	$\rho = \rho'$	Allow 100% of increase
	-0.01	$\rho = \rho'$	Allow 60% of increase
	-0.005	$\rho = \rho'$	Allow 20% of increase
	0	$\rho = \rho'$	Allow 0% of increase
	0.005	$\rho = (1 - 40\%) \times \rho'$	Allow 0% of increase
	0.01	$\rho = (1 - 80\%) \times \rho'$	Allow 0% of increase
	0.015	$\rho = 0$	Allow 0% of increase

The value of $e(t)$ in Rule Table 2.1 is the “center” of the membership function, with “degree of truth” being 1. When $de(e(t))$ is not 1, the effect of the regulation policy should time the value of $mu(e(t))$ in order to reflect the “fuzzy” property of the decision. For example, consider $e(t)$ is 0.003 as it is shown on Figure 2.3. Since the the membership function yields α as the “degree of truth”, the effect of rule 5 is :

$$\rho = mu(\alpha) \times 60\% \times \rho', \text{ with 0\% of increase being allowed.}$$

$mu(\alpha)$ is the truth value of rule 5, which reflects the confidence of that rule.

2.2.3 Interference Unit Design

Determining the applicability of each rule is called “matching”. We say that a rule is “on at time t” if the degree of truth $de(e(t))$ is bigger than 0. Hence, the inference mechanism first seeks to determine which rules are on and to find out which rules are relevant to the current situation. In the next step, the inference mechanism will seek to combine the recommendations of all the rules to come up with a single conclusion.

Another factor affects the application of rules is the network state signal transferred from the network state recognizer. In order to avoid the drawbacks of static regulation, we use the network state signal to tune the effect of the rules. The goal is that for an empty network, the increase of injection rate should be encouraged and the decrease should be discouraged. Similarly, if the network is under heavy congestion, the decrease of injection rate should be encouraged and the increase

should be discouraged. If the network is working in normal state, then no tuning should be made.

Generation of network state signal is the task of network state recognizer. Here, we use K ($K \geq 1$) to symbolize the signal. K is proportional to the degree of the network state. For example, for saturated network, the more the network is saturated, the bigger the K is. The same principle applies to empty network.

Using K , for empty network, the effects of rule 1, 2, and 3 should be encouraged, which is done by timing K ; the effects of rule 5, 6, and 7 should be discouraged, which is done by timing $\frac{1}{K}$. Thus, the rule table of the fuzzy controller for empty network is listed in Table 2.2.

Table 2.2. Rule Table of the Empty Network

<i>Regulation Policy</i>		
	ρ	ρ'
$e(t)$	-0.015 $\rho = \rho'$	Allow $K \times 100$ % of increase
	-0.01 $\rho = \rho'$	Allow $K \times 60$ % of increase
	-0.005 $\rho = \rho'$	Allow $K \times 20$ % of increase
	0 $\rho = \rho'$	Allow 0% of increase
	0.005 $\rho = (1 - 40\% \times \frac{1}{K}) \times \rho'$	Allow 0% of increase
	0.01 $\rho = (1 - 80\% \times \frac{1}{K}) \times \rho'$	Allow 0% of increase
	0.015 $\rho = 0$	Allow 0% of increase

For saturated network, the effects of rule 5, 6, and 7 should be encouraged, which is done by timing K ; the effects of rule 1, 2, and 3 should be discouraged, which is done by timing $\frac{1}{K}$. Thus, the rule table of the fuzzy controller for saturated network becomes Table 2.3.

Table 2.3. Rule Table of the Heavily Saturated Network

<i>Regulation Policy</i>		
	ρ	ρ'
$e(t)$	-0.015 $\rho = \rho'$	Allow $\frac{1}{K} \times 100$ % of increase
	-0.01 $\rho = \rho'$	Allow $\frac{1}{K} \times 60$ % of increase
	-0.005 $\rho = \rho'$	Allow $\frac{1}{K} \times 20$ % of increase
	0 $\rho = \rho'$	Allow 0% of increase
	0.005 $\rho = (1 - 40\% \times K) \times \rho'$	Allow 0% of increase
	0.01 $\rho = (1 - 80\% \times K) \times \rho'$	Allow 0% of increase
	0.015 $\rho = 0$	Allow 0% of increase

For network that is around the saturated point, K is zero and the effects of the rules remains unchanged.

Consider now for one network working in heavily saturated condition, how we compute which rules are on. Suppose that during a period of time, we have three

Figure 2.4. Computation of $\mu(e(t_1))$

samples of one data source's cache missing rate, which are:

$$\begin{aligned}\rho_1 &= 0.037 \text{ packet/cycle, and } e(t_1) = 0.012 \text{ packet/cycle} \\ \rho_2 &= 0.013 \text{ packet/cycle, and } e(t_2) = -0.012 \text{ packet/cycle} \\ \rho_3 &= 0.009 \text{ packet/cycle, and } e(t_3) = -0.016 \text{ packet/cycle}\end{aligned}$$

Membership function in Figure 2.3 indicates that $de(e(t_1))$ is 0.6, with the corresponding “linguistic” information is “High”; $de(e(t_2))$ is 0.6, with the corresponding “linguistic” information is “Low”; $de(e(t_3))$ is 1, with the corresponding “linguistic” information is “Very low”.

$de(e(t_1))$ invokes rule 6, $de(e(t_2))$ invokes rule 2 and $de(e(t_3))$ invokes rule 1. The computation of $\mu(e(t_1))$ is shown in Figure 2.4.

The function on the right side of the y-axis is the consequence function of rule 6 (see [10, p. 43 - 50]). It is with the same shape of the corresponding membership function. The area (or “mass”) of the gray part is $\mu(e(t_1))$. The effect of one rule depends on the value of $\mu(e(t_1))$. For example, the effect of rule 6 with $\mu(e(t_1))$ is:

$$\rho = \mu(e(t_1)) \times (1 - 80\% \times k) \times \rho', \text{ Allow } 0\% \text{ of increase}$$

Remember that we assume the network is under heavy saturation, thus rules in Table 2.3 are used.

Following the same procedures, we can compute $\mu(e(t_2))$ of $de(e(t_2))$, and $\mu(e(t_3))$ of $de(e(t_3))$. The difference is that $de(e(t_2))$ invokes rule 2 and $de(e(t_3))$ invokes rule 1. These results are used by the defuzzification unit to compute the final control output.

2.2.4 Defuzzification Unit Design

In this section, we introduce the defuzzification unit, which is the final component of the fuzzy controller. Defuzzification operates on the implied fuzzy sets produced by the inference mechanism and combines their effects to provide the “most certain” controller output (plant input). Some think of defuzzification as “decoding” the

fuzzy set information produced by the inference process (i.e., the implied fuzzy sets) into numeric fuzzy controller outputs.

In defuzzification, we want to find one output, which we denote by “ u^{output} ”, that best represents the conclusions of the fuzzy controller. There are actually many approaches to achieve defuzzification. Due to its popularity, we use the “centroid” method, which is also called “center of gravity” (COG) defuzzification method, for combining the recommendations represented by the implied fuzzy sets from all the rules. Let b_i denote the center of the membership function’s consequence function (i.e., the full scale effect of each rule). For our example we have the following rules being invoked:

$b_1 : \rho = (1 - 80\% \times K) \times \rho'$, Allow 0% of increase, $b_2 : \rho = \rho'$, Allow $\frac{1}{K} \times 60\%$ of increase, $b_3 : \rho = \rho'$, Allow $\frac{1}{K} \times 100\%$ of increase.

The COG method computes u^{output} to be

$$u^{output} = \frac{\sum_i b_i \mu(t_i)}{\mu(t_i)}. \quad (2.2)$$

Putting each of the rules in Equation (2.2) we get:

$$u^{output_ \rho'} = \frac{\mu(t_1)(1 - 80\% \times K)\rho' + \mu(t_2)\rho' + \mu(t_3)\rho'}{\mu(t_1) + \mu(t_2) + \mu(t_3)} = (1 - 21.8\% \times K)\rho'.$$

And the increase part of ρ' is:

$$u^{output_increase} = \mu(t_1) \times 0\% + \mu(t_2) \times \frac{1}{K} \times 60\% + \mu(t_3) \times 100\% = 136\% \times \frac{1}{K}.$$

Thus, when the network is saturated, and assume $\rho_1 = 0.037 \text{ packet/cycle}$, $\rho_2 = 0.013 \text{ packet/cycle}$, $\rho_3 = 0.009 \text{ packet/cycle}$, the flow regulation policy for the data source is:

$$\rho = (1 - 21.8\% \times k)\rho', \text{ and allow } 136\% \times \frac{1}{K} \text{ of increase of } \rho',$$

where K is the qualification factor of the network congestion situation. The more severe saturation is, the bigger K becomes.

The above fuzzy flow regulation policy will be given to the flow regulator to regulate the flow. The regulation result is that the injection rate will change from ρ' to ρ , and stores the delayed packets in a source queue waiting for transfer.

2.3 Network State Recognizer Design

The network state recognizer is also based on fuzzy logic. Its function is to recognize the state of the network, generate network state factor “ K ”, and pass it to the fuzzy controller. The design procedure of the network state recognizer shares some similarities with the design of the fuzzy controller. However, differences exist in two aspects:

Figure 2.5. Membership Function for Latency **Figure 2.6.** Membership Function for Link Utilization**Table 2.4.** Rule Table of the Network State Recognizer

<i>NetworkState</i>		<i>Latency(cycle)</i>		
<i>s</i>		<i>Low</i>	<i>Middle</i>	<i>High</i>
<i>Link</i>	<i>Low</i>	<i>Empty</i>	<i>Middle</i>	<i>High</i>
<i>utilization</i>	<i>Middle</i>	<i>Middle</i>	<i>Middle</i>	<i>High</i>
<i>(packet/cycle)</i>	<i>High</i>	<i>High</i>	<i>High</i>	<i>High</i>

1. The network state recognizer takes both latency and link utilization as input signals, which means the fuzzy logic has two inputs instead of one.
2. There are one network state recognizer per CMP chip, not per data source. This means all the data sources (L1 cache, L2 cache and memory bank) inside the same CMP chip share one common knowledge about the state of the network.

For unicast cache request, latency is the delay between the issue time of the request and the arrive time of the response. For multicast cache request, latency is the delay between the issue time of the request and the arrive time of the last response. Link utilization is correlated with the system throughput, which is the average number of successfully sent and received packets per cycle. For fuzzification of delay, three membership functions are used, as shown in Figure 2.5. The membership function that with an open end is used to accept too big or too small latencies. For fuzzification of link utilization, we also three membership functions, as shown in Figure 2.6. The parameters of the latency membership functions and link utilization membership functions are measured through the motivation experiment in Chapter 1. For the rule set, we list all possible rules for the fuzzy controller with two inputs through a table, as it is in Table 2.4.

For network state, “Empty” means empty network, “Middle” means normal network and “High” means highly saturated network.

From the table we can see, if any of the two inputs is *Low*, then the state of the network depends on the value of the other input. If any of the inputs is *Middle*, then the network is in either middle or high state. If one of the input is *high*, then the network is highly saturated. Table 2.4 implies nine rules for the whole rule base.

As to the qualification of the rules, we use **and** interference operation. For example, if latency is middle and link utilization is low, then the following rule will be invoked

IF (Latency is Middle) **and** (Utilization is Low) THEN (Network state is Middle).

We use the *minimum* method to interpret the meaning of **and**. If $de(Latency(t))$ is 0.25, $de(Utilization(t))$ is 0.13, then the definition of **and** is:

- *Minimal* : $de(Latency(t)) \text{ and } de(Utilization(t)) = \min\{0.13, 0.25\} = 0.13$,

which means that 0.13 is the degree of truth of the whole composed premise. There are many other interpretations of the **and**, which can be found in [10, p. 40-49]. The meaning of the *minimum* interpretation method is that if we are not very certain about the truth of one statement, how can we be any more certain about the truth of that statement “and” the other statement? Thus, we choose the minimal of certainty of the degree of truth value. Based on the discussion above, we can get membership functions for all the composed premises. There are totally nine composed membership functions, and we give out one in Figure 2.7. The membership function is for middle latency **and** middle link utilization. Thus, the “degree of truth” of middle latency is above 0 only when $70 < latency < 110$. Similarly, the “degree of truth” of middle utilization is above 0 only when $0.1 < utilization < 0.2$.

For defuzzification, we assign a value K to denote the degree of saturation the network is in. K ranges from -10 to $+10$, where -10 is the highest degree of empty, and 10 is the highest degree of saturation. When K is input to the fuzzy controller, only absolute value is used. The defuzzification function is shown on Figure 2.8. Here we draw the three defuzzification functions on the X-axis. Notice that the COG method never produces values in the grey part. Thus, -10 and 10 are assigned to the “center” of “Low” and “High” defuzzification functions, respectively.

Consider the defuzzification progress of the following example. Suppose during a period of T we have samples of packet delay (75 cycles) and link utilization (0.13 packet/cycle). Then the following fuzzification and defuzzification progress will be involved, as it is shown in Figure 2.9.

Figure 2.9 shows that the defuzzification will produce $de(75) = 0.75$ and $de(0.13) = 0.6$ as the degree of truth of the sampled packet delay and link utilization. The **AND** operation will produce 0.6 as the degree of truth of the composed premises. Since latency is “Low” and link utilization is “Middle”, the ultimate state of the network

Figure 2.7. Membership Function for Composed Premise

Figure 2.8. Defuzzification Function

Figure 2.9. Fuzzification and Defuzzification Progress

Figure 2.10. Defuzzification of Three Samples

is slightly congestion due to the corresponding rule, and the truth value is $mu(0.6)$. Thus, using COG defuzzification method, the network state factor K is

$$K = mu(0.6) \times -10$$

Suppose further that we have two more samples of latency and link utilization, which produce truth values mu_2 , mu_3 . We show the previous mu_1 , and mu_2 , mu_3 on Figure 2.10.

From Figure 2.10 we can see, the network is slightly saturated with truth value mu_1 ; normally saturated with truth value mu_2 , and heavily saturated with truth value mu_3 . The overall factor K of network state is calculated using COG method, which is:

$$K = mu_1 \times -10 + mu_2 \times 0 + mu_3 \times 10$$

Figure 2.11. Control Surface for Network State Recognizer

The output value of K is shown on Figure 2.10. It shows that the state of network is normal, thus the fuzzy controller will apply the rule in Table 2.1. The control surface for the network state recognizer is shown in Figure 2.11. Notice that it is impossible that the linear PD controller can achieve a nonlinear control surface

of the shape shown in Figure 2.11, since PD controller is based on linear differential equations. Thus, fuzzy controller provides more accurate control [10].

2.4 Regulator Design

The regulator in our fuzzy control system is the same as the one in Section 1.2 except that the flow regulation policy can be changed dynamically.

Figure 2.12. Regulator in the Fuzzy Regulation System

Figure 2.12 shows the inputs and output of the regulator in our fuzzy regulation system. When the fuzzy controller makes a new flow regulation policy, the updated σ and ρ are written into the flow regulator. Thus, output flow $D(t)$ reflects the effects of the new regulation rule.

2.5 Summary

This chapter details the design of each component in our fuzzy flow regulation system. In this chapter, we first gave out the system architecture as in Figure 2.1. Then we explained how to design the fuzzification function, rule base, interference mechanism and defuzzification function of the fuzzy controller. Due to the similarities of the fuzzy controller and the fuzzy state recognizer, the design of the latter follows basically the same procedures. However, differences exist in that the latter has two inputs and the former has one input.

Chapter 3

Implementation and Experiment Results

In this chapter, we will show how to implement our system. The system is implemented in C++ and all the components are compatible with GEMS.

3.1 System Level Simulation

Design in this thesis is implemented and evaluated using a full system simulator GEMS. The defining property of full system simulation compared to an instruction set simulator is that the former allows real device drivers and operating systems to be run, not just single programs. Thus, the full-system simulation makes it possible to simulate individual computers and networked computer nodes with all their software, from network device drivers to operating systems, network stacks, middleware, servers, and application programs.

Another difference between GEMS and an instruction set simulator is that GEMS is cycle accurate. A cycle-accurate simulator is a computer program that simulates a micro-architecture on a cycle-by-cycle basis. In contrast, an instruction set simulator simulates an instruction set architecture usually faster but not cycle-accurate to a specific implementation of this architecture. GEMS decouples simulation functionality and timing. The aims of GEMS is to implement components (such as processors, cache, and interconnection network) with detailed timing property. Although the approach decouples functional simulation and timing simulation, the functional simulator is still affected by the timing simulator, allowing the system to capture timing dependent effects. For example, the timing model will determine the winner of two processors that are trying to access the same software lock in memory.

In this thesis, modifications have been done to the following components of GEMS:

Caches. GEMS models a hierarchy of caches associated with each single processor, as well as shared caches used in chip multiprocessors (CMPs) and other hierarchical coherence systems. Cache characteristics, such as size and associativity,

are configuration parameters. In our design, the fuzzy controller exists in the cache component since the statistics of each processor’s cache miss rate can be recorded there.

Interconnection networks. The interconnection network is the unified communication substrate used to communicate between cache and memory controllers. A single monolithic interconnection network model is used to simulate all communication, even between controllers that would be on the same chip in a simulated CMP system. As such, all packet/cycle and inter-chip communication is handled as part of the interconnect, although each individual link can have different latency and bandwidth parameters. In our design, the flow regulator exists in the network interface between network and the corresponding data sources.

Chip. GEMS implements chip to integrate different components together and to simulate a real CMP. The architecture and organization of components (such as processors, private caches and shared caches) can be configured by parameters. Several of these chips are connected and form a multicore system. In our design, one network state recognizer exists in each chip. There is no global network state recognizer since that it is unimplementable when the scale of the system increases.

3.2 Implementation Details

In this section, we discuss the implementation details of the fuzzy-control based flow regulation system. The system has been integrated into GEMS and can be scheduled by the global event queue. All the components are developed in C++, and the class diagram is shown in Figure 3.1.

As it is shown in the figure, GEMS system includes two main modules: Garnet (the interconnection network module) and Ruby (the detailed memory module). In this project, the added components are all within Garnet, while the memory module is kept untouched.

After integration, the Garnet network contains two sub-modules: network and regulator. The network module simulates the functionality of the interconnection network. The main components are 5-stage pipelined routers, network interfaces and virtual channels.

The regulator module contains all the components described in Chapter 2. In order to integrate new components into GEMS, registration and initialization of the regulator module has been done during system creation.

The fuzzy controller has been implemented as a class which inherits from class *Consumer* in GEMS. It samples the cache miss rate of the local data source once every sampling window, as it is mentioned in Chapter 1. After an amount of samples, the fuzzy controller calculates the new regulation policy and passes it to the flow

Figure 3.1. System Class Diagram

regulator. In our design, the sampling window is 128 GEMS simulation cycles and the amount of samples is 16.

The four elements of fuzzy controller (fuzzification, inference mechanism, rules and defuzzification) are all implemented within the fuzzy controller class. During system creation, each cache controller object creates and initiates a fuzzy controller object as a private member. Thus, the fuzzy controller can only change the regulation policy of the flow regulator associated with the same data source.

The rules are also defined as private members and get initialized during the system instantiation. Since class fuzzy controller inherits from class *Consumer*, it can reschedule itself in the global event queue. This property is used to implement the sampling period. Function “fuzzification” is rescheduled every sampling period and each time stores the new sample into a vector. After 16 samples, function “inference” and “defuzzification” analyze these samples using the rule base and generate a new flow regulation policy according to network state factor K , which is generated by the network state recognizer.

The network state recognizer does not inherit from class *Consumer* since it needs not to reschedule itself. It is called by the fuzzy controller during the making of new regulation policy. That is, every 2048 (16×128) cycles. The flow regulator does not inherit from class *Consumer*, since it needs not to reschedule itself either. The flow regulator is within every instance of the network interface.

3.3 Experiment Setup

In the experiment phase, measurements of the packets are done with both open-loop and closed-loop methods. The open-loop method focuses on the measurement of the performance of network and flow regulation, while the closed-loop method focuses on measurement of full system performance. In our experiments, the processors are synthetic since in the open-loop measurement we use synthetic flows, and in the closed-loop method we use trace-files. All the other components (L1 cache, L2 cache and memory banks) are detailed models which mimic the timing properties of real hardware. The experiments are carried out using the same CMP as in Figure 1.5.

3.3.1 CMP Architecture Parameters

The architecture of the experiment CMP is shown in Figure 1.5. Each CMP chip includes four processors (each with one L1 cache) and four L2 cache banks. The L1 cache is private and can only be read or wrote by the processor to which it is attached. The L2 cache is shared, which means all the processors within the same chip can access to all the shared L2 caches. In this thesis, we simulate four CMP chips, with 16 L1 cache banks and 16 L2 cache banks totally. Each L1 bank is 64KB in size, and each L2 bank is 4MB.

Outside of the CMP chip exits the external memory module. The memory module mimics a detailed DDR2/DDR3 memory running in closed page mode with

Figure 3.2. Physical Architecture of the Experiment CMP

posted CAS. It models bank busy time, memory bus occupancy and turnaround delays, and refresh. Each instance of the memory controller models one address bus, one data bus, and any number of DIMMs. Each DIMM has a configurable number of ranks of DRAM. The configuration and delay values are parameterized which can be found in *rubyconfig.defaults* file. The total external memory is divided into four banks. Each bank is attached to one CMP chip. Total memory size is 4 GB, with each bank 1 GB. Within each memory bank, the memory page size is 4KB and the data block size is 64 B.

As to the network module, the topology of the interconnection network is mesh, as shown in Figure 3.2. Notice that Figure 1.5 is a conceptional structure, and Figure 3.2 is the real physical connection. Each network link has four virtual channels, where flits of packets are traveling through. There are two basic kinds of packets, one is data packet and the other one is control packet. One data packet consists of five flits, and one control packet consists of one flit. Each flit is 16 bytes and can be transferred via an internal link in one cycle. That means the internal bandwidth is 16 bytes per cycle. As to the external link, 20 cycles are needed to transfer a flit, which yields the bandwidth to 0.8 bytes per cycle.

The routing algorithm is X-Y dimensional routing, and the flow control is credit based virtual channel flow control. The initial credit number is 16. Since the routing

Figure 3.3. Measurement Setup for Open-loop Measurement

algorithm is deterministic, the network is deadlock free. The router is five-stages pipelined router which has been discussed in [6, 306]. The arbitration policy of the virtual channel allocation and switch allocation is round-robin. The routing mechanics is table based routing, by which the routing path is pre-stored in each router. The cache coherence protocol is MOESI CMP directory protocol. It is a two-level directory protocol for Chip-Multiprocessors (CMP). The L1 and L2 controllers are split, and the L2 cache is shared by all processors on the same chip. Inclusion is maintained between L2s and the L1s, and a sharers list is kept in each L2 cache line.

3.3.2 Open-loop Measurement

With the open-loop method, the delayed packets (due to lack of tokens) are queued inside the source queue. The measurement starts before packets entering the source queue, and ends after the response packets leaving the network interface. The measurement setup of the delay and link utilization is shown in Figure 3.3. In the figure, the packet source can be a L1 cache controller, a L2 cache controller or a memory bank controller. We further assume that the source queue is of infinite size. The packet source initiates a memory transaction to fetch data from or send data to the corresponding component(s) through the network.

The source queue in Figure 3.3 enables the traffic parameters to be controlled independently of the network itself. Without the source queue, a packet source may attempt to inject a packet at a time when the network is unable to accept traffic - for example, when the network's input buffers are full. In such a case, the traffic produced by the source is influenced by the network and is not the traffic pattern originally specified.

Since the source queue separates the network and the data source, flow ρ follows strictly the pattern defined by the packet source. In our experiments, we apply two traffic patterns to our network. One is uniform random traffic, and the other one is bit permutation traffic. Uniform random traffic is used to simulate the situation

that the network loads are perfectly balanced. In reality, network may have different stress in different area, and using uniform random traffic only can not cover this case. To stress the network with unbalanced traffic, bit permutation traffic is used. In bit permutation traffic, each source sends all of its packets to a fixed destination.

Figure 3.4. Traffic Patterns for Bit Permutation

The patterns of the bit permutation flows we use are shown in Figure 3.4. Each line in the figure denotes one flow between the source and the destination, and the flow can travel in either direction. Notice that the lines are not the paths of the flows. The source and destination are center symmetric, which can be expressed by the following formula:

$$d_i = num_nodes + 1 - s_i$$

where d_i is the destination for data source s_i , and num_nodes is the total number of data sources in the system. From the figure we can see the “center” of the bit permutation flows clearly. But notice that the “center” is not a node in the system. In the system that was introduced in Figure 1.5, we have 16 cores (each core is integrated with one L1 cache), 16 L2 caches and 4 memory banks. Thus the total number of data sources in the system are 36, as shown in Figure 3.2 and in Figure 3.4.

In open-loop measurement, the injection of the data source is controlled by Poisson process. In probability theory, a Poisson process is a stochastic process which counts the number of events and the time that these events occur in a given time interval. The time between each pair of consecutive events has an exponential distribution with parameter λ and each of these inter-arrival times is assumed to be independent of other inter-arrival times.

Formally, a Poisson process can be expressed by formula 3.1:

Figure 3.5. Measurement Setup for Closed-loop Measurement

$$P[(N(t + \tau) - N(t)) = k] = \frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!} \quad k = 0, 1, \dots \quad (3.1)$$

where $N(t + \tau) - N(t) = k$ is the number of events in time interval $(t, t + \tau]$. Thus, the average injection rate r_{avg} during interval τ is:

$$r_{avg} = \frac{E(P(N(t + \tau) - N(t)) \times k)}{\tau} = \frac{E(\frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!} \times k)}{\tau} = \frac{\lambda\tau}{\tau} = \lambda$$

In our experiment, τ is set to the length of 256 sample windows. Within the same τ , injection rate is kept as a constant.

3.3.3 Closed-loop Measurement

Unlike the open-loop method, the closed-loop method does not insert a source queue between the packet source and the network interface. The structure of the closed-loop measurement setup is illustrated in Figure 3.5.

Since the packet source interacts with the network, the injection rate doesn't conform strictly to the traffic pattern. For example, the packet source will stop injecting packet if the network can not accept additional packets due to saturation. The closed-loop method is used to test the sensitivity of the application run time on network parameters such as bandwidth, routing algorithm, and flow control.

In order to test the fuzzy-based flow regulator under real applications, we use trace file as the packet source in the closed-loop measurement. The trace file is generated by running test bench Splash2 on Simics with GEMS. It records the cache (both L1 and L2 cache) and memory access requests during the running of the Splash2 programs.

Each line of the trace file contains the following information: simulation time, packet source, packet destination, packet access type (data or control). If the packet is a data packet, the trace file also contains additional information such as the

Figure 3.6. Accumulative Packets Curve

access mode (write, read), address, data and some cache coherence protocol specified information (for example, information to indicate an atom read/write operation). If the packet is a control packet, the trace file shows the type of control, destination(s) of the control, and the entry in a remote cache directory. As mentioned before, we assume that all control packets are one flit and all data packets are five flits, with one flit consisting of 16 bytes.

In the next section, we report the experiment results, with open-loop method first then closed loop method.

3.4 Experiment Results

3.4.1 Results of Open-loop Method with Uniform Random Traffic

In the open-loop measurement, we assume that each data source has the same injection rate. Notice that although the injection rate is identical, each packet source has different Poisson process. This section shows the results of the experiments using uniform traffic pattern as input.

Figure 3.6 shows the accumulative packets over time of one processor, which reflects the property of flow ρ in Figure 3.3. The injection rate is 0.012 packet/cycle at cycle 0, then increases by 0.001 packet/cycle during each timing window (65536 cycles). Due to the increase of injection rate, the slope of the accumulation curve increases, as shown in the figure.

Without fuzzy-based flow regulator, static regulated flow or unregulated flow “blindly” injects packets into the network. This is due to the fact that static or unregulated flow does not know the state of the network. It keeps injecting packets until the network is totally saturated. The accumulative packets curves for static regulated and unregulated flows are showed in Figure 3.7.

Figure 3.7. Accumulative Packets for the Four Flows

Due to the limitation of space, Figure 3.7 is drawn from cycle 73560 to cycle 73680, approximately. One data packet is transferred at cycle 73570. Remember that one data packet consists of 5 flits, which are transferred in a bursty fashion.

Unregulated flow follows the source flow closely, because there is no regulation policy. Static regulated flow keeps the scope as a constant which reflects the “static” property of the regulation policy.

Being different, the fuzzy-control based regulation policy regulates flows according to the network state. For example, on encountering the burst at cycle 73570, the fuzzy controller first recognizes the network state through the fuzzy network state recognizer. On knowing the network state is near saturation (the injection rate is 0.016 packet/cycle, which is very near the reference injection rate 0.02 packet/cycle), the fuzzy controller “tightens” the regulation policy up, as it is showed in Figure 3.7.

Figure 3.8 shows the injection rate over time of the source flow, unregulated flow, statically regulated flow, and fuzzy regulated flow. Since the unregulated flow follows the source flow very closely, they share the same injection rate curve. When the injection rate is under 0.02 packet/cycle (which is the ρ parameter of static regulator), the static regulated flow has more or less the same rate with the source flow. But when the injection rate is beyond 0.02 packet/cycle, the static regulated flow keeps 0.02 packet/cycle as the static injection rate. The result is that the packets that can be accepted by the network are postponed in the source queue, which causes higher queuing delay.

In order to show the regulation effects of fuzzy-based regulator, we magnify the rectangle in Figure 3.8 and re-draw it in Figure 3.9. The dashed line is for fuzzy regulated flow. The solid line is for both source flow and static regulated flow.

Figure 3.9 tells us that when the network is empty (as it is denoted by “E”), the fuzzy regulated flow injects more packets than unregulated flow. Thus, over-regulation is avoided and the network resources are more fully used. When the

Figure 3.8. Injection Rate for the Four Flows

Figure 3.9. Effects of Fuzzy Controller for Uniform Random Traffic

network is working in normal state (as it is denoted by “N”), the fuzzy regulated flow and the unregulated flow have the same injection rate. When the network is saturated (as it is denoted by “S”), the fuzzy regulated flow has a lower injection rate, which means that under-regulation is addressed and the network starts to cover from saturation.

Notice that the horizontal gap between a regulated flow and the source flow in Figure 3.7 manifests the queueing delay of one packet due to lack of tokens. It is equal to the time one packet stays in the source queue. From the figure we can say static regulation has the biggest queueing delay and unregulated flow has the smallest. Queueing delay of the fuzzy regulated flow lays in the middle.

The vertical gap between a regulated flow and the source flow manifests the

buffer usage of the source queue. Given a fixed time point, the gap is equal to the amount of packets waiting in the source queue. From the figure we can say static regulation has the biggest buffer usage while unregulated flow has the smallest. Usage of the fuzzy regulated flow again lays in the middle.

As to more meaning about the horizontal gap and the vertical gap, please refer to [8, p. 24 - 25].

Figure 3.10 shows the histograms of different packet delays using static flow regulation. The delay includes total delay, queueing delay and network delay. Queueing delay is the time that a packet waits for network resources in the source queue. Network delay is the network transaction time from data source to destination. The average value for total delay is 141.57 cycles, for queueing delay is 110.4524 cycles, and for network delay is 37.6758 cycles.

Figure 3.11 shows the histograms of different packet delays of an unregulated flow. The average value for total delay is 83.6755 cycles, for queueing delay is 3.0057 cycles, and for network delay is 79.8787 cycles.

Figure 3.12 shows the histograms of different delays of a fuzzy regulated flow. The average value for total delay is 73.1232 cycles, for queueing delay is 28.5141 cycles, and for network delay is 49.8101 cycles.

Table 3.1. Delays from Different Regulation Policies

Average Delay (cycles)	Regulation Policy		
	No regulation	Static regulation	Fuzzy regulation
Queueing Delay	3.0057	110.4524	28.5141
Network Delay	79.8787	37.6758	49.8101
Total Delay	83.6755	141.5706	73.1232
Simulation Time	1000000 Cycles		

Table 3.1 summarizes the delays of the three regulation policies. Without regulation policy delaying packets in the source queue, the unregulated flow has the lowest queueing delay. The consequence is that it has the highest network delay. The static regulation policy works fine when data source injects packets slower than 0.02 packet/cycle. When the injection rate increases, queueing delay increases dramatically since more packets are waiting in the source queue. From the table we can see the average total delay of static regulated flow is bigger than that of unregulated flow. This tells us that statically regulation policy successfully avoids network congestion when injection rate is more than the acceptable value, but the cost is the increase of average total packet delay.

With the fuzzy-control based regulation policy, both network delay and queueing delay fall in the middle. The regulated flow does not encounter a high queueing delay because that the flow regulator encourages the raising of injection rate when the network is empty. It does not encounter high network delay either since the flow regulator forbids additional packets entering the network when saturation happens. The consequence is that the total delay, as the summation of the queueing delay

Figure 3.10. Delay Histograms for Static Regulated Flows

Figure 3.11. Delay Histograms for Unregulated Flows

Figure 3.12. Delay Histograms for Fuzzy Regulated Flows

Figure 3.13. Link Utilization for Regulated Flows

and the network delay, reaches the lowest value of the three flows.

Figure 3.13 shows the average link utilization curves of the three regulated flows. The curve of unregulated flow increases from 0.11 packets per cycle to 0.17 packets per cycle. The curve of fuzzy regulated flow is close to the curve of unregulated flow, but never exceeds it. This is because the fuzzy flow regulator tries to make the best usage of the network resources, but the usage can not be higher than that of an unregulated flow. The unregulated flow injects packets as soon as possible, and the network has the highest link utilization.

Link utilization curve of the static regulated flow begins to slow down when injection rate exceeds 0.02 packet/cycle. This is because additional packets are kept in the source queue and the network does not transfer more packets when it actually can.

3.4.2 Results of Open-loop Method with Bit Permutation Traffic

In this section, we show the experiment results with bit permutation traffic. As it is discussed in Section 3.3.2, uniform random traffic tests the network assuming that the work load is balanced, and bit permutation traffic stresses the network with unbalanced work load. Thus, we expect that the network is easier to be saturated using permutation traffic. More about bit permutation traffic can be found in [6].

The traffic injection process is also Poisson process. The injection rate increases from 0.012 packet/cycle to 0.028 packet/cycle, with the increase step of 0.001 packet/cycle every 8192 cycles.

Figure 3.14 shows the injection rate of the source flow, unregulated flow and fuzzy regulated flow. Comparing with Figure 3.9, the effects of fuzzy regulation are more obvious. This is due to the fact that the network becomes more saturated

Figure 3.14. Effects of Fuzzy Controller for Bit Permutation Traffic

with bit permutation traffic than with uniform random traffic. Figure 3.15 shows the histograms of different delays of static regulated flows. The average value for total delay is 127.6505 cycles, for queueing delay is 100.4710 cycles, and for network delay is 34.2457 cycles. Because bit permutation traffic has fixed source-destination pairs, the network delay does not span to all the values in the X axis. Figure 3.16 shows the histograms of different delays of unregulated flows. The average total delay is 95.9200 cycles, average queueing delay is 5.6969 cycles, and average network delay is 87.0675 cycles. Figure 3.17 shows the histograms of different delays of fuzzy regulated flows. The average total delay is 59.9684 cycles, average queueing delay is 27.3868 cycles, and average network delay is 95.9200 cycles.

Table 3.2. Delays from Different Regulation Policies

Average Delay (cycles)	Regulation Policy		
	No regulation	Static regulation	Fuzzy regulation
Queueing Delay	5.6969	100.4710	27.3868
Network Delay	87.0675	34.2457	40.9297
Total Delay	95.9200	127.6505	59.9684
Simulation Time			1000000 Cycles

Table 3.2 summarizes the delays in Figure 3.15, Figure 3.16, and Figure 3.17. The trend in Table 3.2 is the same as in Table 3.1. Unregulated flows have the lowest queueing delay, due to the fact that no flow regulator delays packets in the source queue. As a consequence, the network delay is the highest since the network has the highest work load.

Static regulated flows have the highest queueing delay. The static regulator blindly delays packets in the source queue when the injection rate of data source exceeds the reference value. The benefit is that the network always avoids satura-

Figure 3.15. Delay Histograms for Static Regulated Flows

Figure 3.16. Delay Histograms for Unregulated Flows

Figure 3.17. Delay Histograms for Fuzzy Regulated Flows

Figure 3.18. Link Utilization for Regulated Flows

tion, and the lowest network delay is reached. The cost is that the queueing delay is dramatically increased.

From the table we can see that with fuzzy flow regulation policy, the average total delay reaches the lowest value. This is because the fuzzy regulator makes regulation decision based on the network state. The fuzzy flow regulation policy does not pass too much work load to the network as the unregulated flow does, nor does it hold packets in the source queue for unnecessary cycles as the static regulation policy does. The fuzzy regulator balances the queueing delay and network delay dynamically, according to the network state.

Figure 3.18 shows the link utilization of the network under no regulation policy, static regulation policy and fuzzy regulation policy. Fuzzy controller also makes good link utilization under bit permutation flows, since the curve of fuzzy regulation policy closely follows the curve of unregulation policy.

3.4.3 Discussion of Open-loop Results

This section compares the results in section 3.4.1 and 3.4.2. In this section, we focus on the average total delay of the three regulation policies.

Figure 3.19 compares the total delay histogram of the three regulation policies, both for uniform random traffic and bit permutation traffic.

The figure shows that the static regulated flows have a large distribution in delay range from 100 cycles to 500 cycles, which is due to the queueing delay in the source queue. The unregulated flows have the narrowest delay span (from 50 cycles to 250 cycles approximately), but 95% of the packet delays fall in rang from 70 to 100 cycles, which is caused by the high network delay. The delay of fuzzy regulated flow spans from 5 cycles to 300 cycles, and the distribution span focuses on 50-150

Figure 3.19. Delay Histogram Comparison

cycles. This shows that the delay of fuzzy regulated flow does not have a large distribution in 100 cycles to 500 cycles, which is caused by unnecessary queueing. Nor does it has a large distribution in 70 to 100 cycles, which is caused by high network delay.

Figure 3.20. Composition of Delays

Figure 3.20 gives the composition of the total delay. The figure straightforwardly shows that the queueing and network delays of fuzzy regulated flows do not reach the lowest value. But as a summation, the total delay reaches the lowest value of the

Figure 3.21. Effects of the Regulation Policies on Different Injection Rates

three. Thus, the fuzzy regulation policy improves the performance of the network, both with uniform random traffic and with bit permutation traffic.

Figure 3.21 shows the effects of the three regulation policies on different network states. The network state is denoted by the K factor which is generated by the network state recognizer. When $-10 < K < -5$, the network is empty. When $-5 < K < 5$, the network is normal. And when $5 < K < 10$ the network is saturated. The upper figure is for uniform random traffic, and the lower figure is for bit permutation traffic. When the network is in empty state, all of the three policies yield similar results (45 cycles for total delay). This delay is close to the zero load delay of the network. This phenomenon shows when network is empty, none of the three regulation policies makes obvious effect and all of the network flows are unregulated. When network state is near the saturation point, fuzzy regulation and no regulation beat static regulation policy. This is due to the fact that static regulation policy guarantees a low network delay, with the cost of high queueing delay. When network is in saturated state, the fuzzy regulation policy gets the lowest total delay of the three. The figure illustrates that no matter what state is the network in, the fuzzy regulation policy always makes good regulation decisions.

3.4.4 Results of Closed-loop Method

In closed-loop method, we use network throughput as the indicator of performance. There is no source queue between the packet source and the network interface, and delay of packets can not be divided into queueing delay and network delay.

The benchmark traces are generated from running Splash2 on Simics with

Figure 3.22. Results of Closed-loop Measurement

GEMS, with the same configuration as in Figure 1.5. The traces record the cache and memory access requests of each data source. There are totally 14 benchmark programs and we use 4 of them as our testing traces.

The 4 programs we use are: FFT, water, barnes and ocean. The FFT is used to calculate Fast Fourier Transform. The water is an application that evaluates the forces in a cluster of water molecules in liquid state. The barnes simulates the interaction of a system of bodies (galaxies or particles, for example) in three dimensions over a number of time-steps. The ocean calculates large-scale ocean movements based on eddy and boundary currents.

Figure 3.22 shows that the throughputs of the network are different case by case. The FFT program has the lowest throughput while the Ocean has the highest. Throughput reflects the network usage by each data source, and the network usage is dependent on the cache miss rate of each core. The figure implies that the FFT has the lowest cache miss rate, and the Ocean has the highest.

In benchmark FFT, the improvement of fuzzy regulation against no regulation is 0.0034 packet/cycle (9.39%), and against static regulation is 0.0088 packet/cycle (28.5%). In benchmark Water, the improvement of fuzzy regulation against no regulation is 0.0082 packet/cycle (11.7%); and against static regulation is 0.0181 packet/cycle (31.0%). In benchmark Barnes, the improvement of fuzzy regulation against no regulation is 0.0453 packet/cycle (25.7%), and against static regulation is 0.0641 packet/cycle (41.0%). In benchmark Ocean, the improvement of fuzzy regulation against no regulation is 0.0519 packet/cycle (25.0%), and against static regulation is 0.0737 packet/cycle (39.7%).

The effects of the fuzzy regulation become more obvious as the cache miss rate

increases. This has previously showed in Figure 3.21. Figure 3.22 also shows that within the same benchmark program, the fuzzy regulation yields the best throughput and the static regulation yields the worst. This is consistent to our observations in Section 3.4.1 and Section 3.4.2.

3.5 Summary

In this chapter, we report experiment results for our fuzzy flow regulation system. The experiments were done with both the open-loop and the closed-loop methods. In the open-loop measurement, we compared total delay, queueing delay and network delay of three different flow regulation methods: no regulation, static regulation and fuzzy regulation. The results manifested that the fuzzy regulated flow always had the lowest total delay. In the closed-loop measurement, we used network throughput as the metric of performance. We also compared the throughput with the three different flow regulation policies. The results show that network under fuzzy flow regulation always has the highest throughput.

Chapter 4

Conclusions and Future Works

4.1 Conclusions

The results from the open-loop measurement and the closed-loop measurement jointly show that our fuzzy based regulation policy achieves better performance than static regulation policy and no regulation policy. The busier the network is, the more obvious the effects are.

As to the open-loop measurement, although fuzzy regulation does not guarantee the lowest queueing delay nor the lowest network delay, it guarantees the lowest total delay. The fuzzy regulator adjusts the regulation policy according to the network state. When the network is empty, the fuzzy regulator “loosens” the regulation policy and transfers more packets than static regulation policy. Thus, packets avoid unnecessary queueing delay in the source queue.

When the network is saturated, the fuzzy regulator “tightens up” the regulation policy and holds back packets in the source queue. With this behavior, the fuzzy regulator does not stress the network with too much workloads under network saturation.

The link utilization of the network also gets good results with the fuzzy regulation policy. The utilization value is much higher than the value of static policy, and very close to the value of no regulation policy.

In the closed-loop measurement, the data sources and the network are integrated together and there are no source queues lay in between. Thus delays can not be categorized into queueing delay and network delay. In the closed-loop measurement, we use throughput of the whole network as indicator of the network performance.

Results in Figure 3.22 shows that network with fuzzy regulation reaches the highest throughput. The effects of fuzzy regulation on throughput become more obvious as the cache miss rate increases. Figure 3.22 also shows that within the same benchmark program, fuzzy regulation always yields the best result.

4.2 Further Works

Although this thesis shows that network with fuzzy regulation policy guarantees good performance, there are still questions waiting to be explored. We list some of them in the following.

1. This thesis has compared fuzzy regulation with no regulation and static regulation. Another interesting and meaningful comparison would be between fuzzy regulation and dynamic regulation. About work on dynamic flow regulation, please refer to [1].
2. The experiments (both open-loop experiments and closed-loop experiments) in this thesis are done with Splash2 as the benchmarks. In the future, other benchmark programs can be used (for example, PARSEC) for testing the sustainability of the fuzzy controller.
3. The philosophy behind this thesis is that the fuzzy controller mimics the behavior of a control “expert” between the controller and the NoC. The control decisions are made through experience and ad-hoc methods. As an expansion of this thesis, formal theory for the NoC behavior under control waits to be explored.
4. Due to time limitations, we did not run operating system with the experiments. The experiments are done by using trace files. In the future, testing can be carried out with an operating system booted in order to simulate a more accurate CMP environment.
5. All the experiments in this thesis are setup with the same configuration. That is to say, 16 cores connected by mesh interconnection network. Other configurations can also be used in the future. For example, 64 cores with 64 L1 caches and 64 L2 caches connected by a torus network. Dynamic routing is also a good alternative of X-Y dimension routing. Expectation is that more cores will cast more stress on the network, and the regulation results will be more eminent.
6. One of the most important future works is that the hardware cost and overhead of our fuzzy regulation system should be estimated. In the future, the fuzzy regulator should be integrated into a real hardware system for the testing of effectiveness.
7. Since this thesis is implemented in pure software environment, the overhead and delay of the fuzzy flow control system are ignored. In real hardware environment, how fast the network state recognizer, fuzzy controller and the regulator can run waits to be estimated. Will the fuzzy control system be the system bottleneck also deserves to be researched.

8. The statistics information such as packets delay and link utilization are obtained from the statistics unit inside GEMS. In real hardware, methods about how to gather these information efficiently with minimal hardware cost should be developed.
9. The sampling window in this thesis is fixed. We believe that the length of the sampling window can also bring magnificent effects into system performance. This aspect should also be researched in the future.

Appendix A

Gems Configuration List

These parameters can be found in file *Ruby/config/rubyconfig.defaults* of the GEMS dispatch[5]. They can be changed according to different design specifications. All the changes can be done both in compile-time or run-time.

The parameters listed here are critical for understanding the configuration of the experimental CMP. Many other parameters, such as the parameters for the detailed memory module, are omitted due to space problem. For a detailed GEMS parameters list, please see the corresponding file in *Ruby/config* of the GEMS dispatch.

Table A.1: Parameters for GEMS

Configuration Parameters	Meaning	Value
SIMICS_RUBY_MULTIPLIER	Determines how many Simics cycles advance for every Ruby cycle	4
SEQUENCER_CONTROLLER_LATENCY	Ruby cycles between when a sequencer issues a request and it arrives at the L1 cache controller	4
SEQUENCER_OUTSTANDING_REQUESTS	Maximum number of requests (including SW prefetches) outstanding from the sequencer (Note: this also include items buffered in the store buffer)	16
g_MEMORY_SIZE_BYTES	Memory size in bytes	4294967296
g_DATA_BLOCK_BYTES	Memory data block size in bytes	64
g_PAGE_SIZE_BYTES	Memory page size in bytes	4096
g_REPLACEMENT_POLICY	Cache replacement policy	LRU
g_NUM_PROCESSORS	Number of processors totally	16

Table A.1: Parameters for GEMS

Configuration Parameters	Meaning	Value
g_PROCS_PER_CHIP	Number of processors per CMP chip	4
g_NUM_L2_BANKS	Number of shared L2 cache banks totally (divided equally among CMP chips)	16
g_NUM_MEMORIES	Number of shared memory banks totally (divided equally among CMP chips)	4
TSO	Total store ordering processor	false
g_CACHE_DESIGN	Cache organization structure	NUMA
g_FLIT_SIZE	Network packet flit size in bytes	16
g_VCS_PER_CLASS	Virtual channels per channel	4
int_node_link_latency	Latency for network channel	1 (cycles)
ext_node_link_latency	Latency for external memory channel	20 (cycles)

Bibliography

- [1] Z. Lu, D. Barchos, and A. Jantsch, “A flow regulator for on-chip communication,” *The 22nd IEEE International SoC conference(SoCC’09)*, 2009.
- [2] J. Zhang, “Design and implementation of axi-based network-on-chip system for flow regulation,” Master’s thesis, The Royal Institute of Technology(KTH), Kista, Sep 2009.
- [3] Y. Wang, “Design and Implementation of Dynamic Flow Regulation for On-chip Networks,” Master’s thesis, The Royal Institute of Technology(KTH), Kista, Sep 2011.
- [4] Z. Lu, Y. Wang, “Dynamic Flow Regulation for IP Integration on Network-on-Chip,” *Sixth IEEE/ACM International Symposium on Networks-on-Chip*, 2012.
- [5] Milo M. K. Martin, et al., “Multifacet’s General Execution-driven Multiprocessor Simulator (GEMS) Toolset,” *Appears in Computer Architecture News (CAN)*, September 2005.
- [6] W. J. Dally, and R. Towles, “Principles and Practices of Interconnection Networks,” Morgan Kaufmann, 2004.
- [7] R. Srikant, “The Mathematics of Internet Congestion Control (Systems and Control: Foundations and Applications),” Springer, 2004.
- [8] J. Boudec, P. Thiran, “Network Calculus - A Theory of Deterministic Queuing Systems for the Internet,” Springer, 2011.
- [9] A. Kumar, D. Manjunath, J. Kuri, “Communication Networking - An analytical approach,” Morgan Kaufmann, 2004.
- [10] K. M. Passino, S. Yurkovich, “Fuzzy Control,” Addison-Wesley, 1997.
- [11] Frank Kelly, Thomas Voice, “Stability of end-to-end algorithms for joint routing and rate control,” *Computer Communication Review* 35:2 (2005) 5-12, 2005.
- [12] J. L. Hennessy, D. A. Patterson, “Computer Architecture: A Quantitative Approach, Fifth Edition,” Morgan Kaufmann, 2010.