MULTIMEDIA COMMUNICATION WITH CLUSTER COMPUTING AND WIRELESS WCDMA NETWORK

By

JU WANG

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2003

This work is dedicated to my beloved wife Qingyan Zhang.

ACKNOWLEDGEMENTS

I would like to thank my dissertation advisor, Dr. Jonathan Liu, for his guidance and support, without which it would never have been possible to complete this work. Dr. Liu's invaluable advice on both academic and nonacademic life made me more mature and scholastic. I would also thank Dr. Randy Chow and Dr. Jih-kwon Peir for sharing their intelligence and experience with me. Thanks also go to Dr. Sahni and Dr. Fang, who provide many constructive questions and witty suggestions on my research work.

Special thanks go to my parents. Their continued support has always provided me the courage and the spirit to move forward. Finally, words will never be enough to express my gratitude to my beloved wife, Qingyan, for her constant love, patience, encouragement, and inspiration during the difficult times.

TABLE OF CONTENTS

Page

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

MULTIMEDIA COMMUNICATION WITH CLUSTER COMPUTING AND WIRELESS WCDMA NETWORK

By

Ju Wang

August 2003

Chairman: Jonathan C.L. Liu
Major Department: Computer and Information Science and Engineering

Distributed multimedia is a multidisciplinary area of research which involves networking, video/audio processing, storaging and high performance computing. Recent advances in video compression and networking technology has brought increasing attention to this area. Nevertheless, providing high-quality digital video in large scale remains a challenging task.

The primary focus of this dissertation is twofold:(1) We investigated a pure-software based parallel MPEG-2 decompression scheme. To achieve a high level of scalability, we proposed the data pipeline scheme based on a master/slave architecture. The proposed scheme is very efficient due to the low overhead in the master and slave nodes. Our experimental results showed that the proposed parallel algorithm can deliver a close-linear speedup for high quality compressed video. With a 100-Mbps network, the 30-fps decompression frame rate can be approached using Linux cluster. With 680-Mbps SMP environment, we observed the 60-fps HDTV quality with 13 nodes. (2) The second topic of this dissertation addressed how multimedia

applications can be effectively supported in the wireless wideband CDMA network. The major challenge lies in the fact that multimedia traffic usually demands high data-rate and low bit error, while the wireless network often experiences high bit error caused by multi-user interference over the air medium. Based on the observation that better channel quality could be obtained by using longer spreading codes, a new media access control (MAC) scheme was proposed such that the spreading code for each mobile is dynamically adjusted based on the network load and desired traffic QoS. With this new MAC protocol, the radio resource can be better utilized and more data traffic can be supported.

We further extend the dynamic spreading scheme to the multi-cell scenario to handle the mobile handover. Based on the system load of local and neighbor cells, the new handoff algorithm will decide when and what spreading factors should be used for a mobile during the handoff period. The algorithm also optimizes the assignment of spreading codes and mobile transmitting power such that the overall throughput can be maximized. The new handoff algorithm is also optimized to allow batch-processing for mobile requests to reduce the handoff delay at heavy traffic.

CHAPTER 1
INTRODUCTION

Over the past decade, a tremendous amount of research and development effort has been undertaken on high performance distributed multimedia systems. Digital video with decent quality (e.g., DVD quality) has already become affordable to the general public with off-the-shelf workstations and various electronic consumer products. Some important multimedia applications, such as high quality video conferencing and video-on-demand, are becoming realistic over wireline network. Nevertheless, it is only human nature to seek even higher video quality (e.g., HDTV) and ubiquitous access to multimedia information. To this end, it is essential that the following two key issues be resolved: (1) a high performance, generic yet scalable software video encoding/decoding solution that does not rely on any particular hardware design and (2) a high performance mobile communication network that can support a large-scale multimedia traffic. In this dissertation, we proposed and investigated various technologies to address these issues. The primary focus of this dissertation is thus twofold: (1) to investigate the design issues of a software-based, parallel MPEG-2 decoder (MPEG-2 is a widely used video format) and (2) to evaluate a new wideband CDMA communication protocol which supports multimedia traffic and significantly increases the link-level QoS for the uplink channel and system performance during the mobile handoff.

1.1 High Performance MPEG-2 Decoding

MPEG-2 standard has been widely accepted as a platform to provide broadcasting quality digital video. Using a powerful DCT transform based compression

algorithm and the motion compensation technique, great compression ratio can be obtained while preserving good video quality. The technique, however, requires intensive processing during the encoding and decoding phase. The computation requirement in the decoding side is particularly critical, because the decoder must be fast enough to maintain a continuous play back.

Real-time MPEG-2 decoding has been implemented via a hybrid approach (general purpose processor with multimedia extension) for the broadcasting-level video quality. Nevertheless, high-performance, scalable, portable software decoder schemes for the high-level video quality are still under investigation. We investigated how a generic high-performance software decoder can be constructed by parallelizing the decoding process over a workstation cluster or multiple processors in an SMP machine. Two parallel approaches, namely data-partition scheme [1] and data pipeline scheme [2], are studied in this dissertation.

With the data-partition scheme, a dedicated master node is in charge of parsing the raw MPEG-2 bitstream (either from a network in the case of streaming video or from a local file system), dividing each frame into sets of macroblocks, distributing subtasks, and collecting the results. In the slave nodes, each macroblock goes through VLC decoding, IDCT, and MC to produce the pixel information. This scheme requires the transmission of reference data when decoding non-intra coded macroblocks (in P- and B-type frames). Our analysis reveals that the communication cost related to reference data can vary significantly with different partition methods. Thus an optimal partition should be found to maximize system performance. We compared four partition schemes based on the reference data communication pattern. The Quick-Partition scheme produced the least communication overhead. Our Quick-Partition algorithm subdivides a given frame into two parts every time, along the shorter dimension. Then the two subparts are further divided using the same strategy. This

procedure repeats until a desired number is achieved. With Quick partition, our parallel decoder can provide a peak decoding rate of 44-fps, under a 15-node SMP system with measured inter-node bandwidth of 680Mbps. The corresponding speedup[1] is about 8.

However, it is found that the data partition method will cause high computation overhead at the master node when the number of partitions becomes large, which results in a low speedup gain. To achieve high system performance at large scale parallel configuration, we investigated the second parallel approach, the data pipeline scheme. The data pipeline scheme is more scalable than the data-partition scheme by using a frame level parallelization. By doing this, not only is the computation overhead in the master node reduced, the communication cost is also significantly reduced, where the cost of transferring reference data can be virtually eliminated.

The performance of our pipeline algorithm is determined by two design factors: (1) the block size D (i.e., the number of frames decoded in each slave) and (2) the number of slave nodes N. The determination of the optimal values of these two factors becomes nontrivial due to the inter-frame data dependence among I-, P- and B-frames. The internal data dependence causes the transmission of reference picture data among different nodes and even limits the degree of parallelism. We find that increasing D can significantly reduce the communication to the reference picture. When N is fixed, this reduction is close-linearly until D equals the size of GOP, where the minimum communication overhead is achieved. Nevertheless, as the performance analysis model will demonstrate, network bandwidth turns out to be a critical factor for the overall system performance. We have found that there is a single saturation point for a given multiple-node computation environment. Before the saturation point, the system round time (i.e., overall time for one pipeline running cycle) is

[1]The speed-up gain is defined as the ratio of decompression time between the parallel decoder and its corresponding serial version. Since the majority decompression work is done in slave nodes, we exclude the master node when calculating the speed-up. This is, the maximum speed-up for an 8-node-configuration (1 master node and 7 slave nodes) will be 7. We use this definition throughout all the discussion unless explicitly stated.

dominated by the CPU processing time and increasing N can increase the system performance close-linear. However, a higher decoding rate also causes more network traffic. Eventually the growing communication traffic will saturate the system when it reaches the system bandwidth limitation.

Experiments over different networking and OS platforms were performed to validate our proposed data pipeline scheme. The actual decompression rate is recorded for these experiments, with a particular focus on 30-fps real time video and 60-fps HDTV quality. The experimental results indicate that our scheme can provide a real-time decoding rate and the system can be scaled up with the 100-Mbps and the 680-Mbps environments. With a 100-Mbps network, we are able to deliver 30-fps with 6 slave nodes (each node can decompress at 5.6-fps), with speed-up of 5.7.

With the 680-Mbps SMP environment equipped with 15 low end processors (Sparc 248MHz), a rate of 30-fps was achieved when using 7 nodes (single node decompression rate of 5-fps). A rate of 60-fps (HDTV quality) was achieved when using 13 and 14 nodes. Our analysis shows that a 270-fps decoding rate could be expected with a full configuration with our SMP testing environment.

In order to evaluate the scalability performance of the data pipeline parallel algorithm, further experiment were conducted with high resolution video format [3]. However, when attempting to decode (1024*1024) and (1404*960) high-level MPEG-2 video, we have observed a severe performance degradation (e.g., dropping from 18 or 20 fps to 2.5 fps) when more than 10 slave nodes are used. By analyzing the runtime system resource utilization, we found that the system memory is quickly exhausted when increasing the number of slave nodes. When decoding the video file with high spatial resolution, the increase of memory usage eventually becomes a system bottleneck. To address the challenge and obtain high scalable decoding for high resolution video, we proposed and implemented two revised memory management approaches to reduce the buffer requirement. The first is Minimum Transmission

Buffer in Slave Node (**ST scheme**), where we reduce the transmission buffer size in the slave nodes to three frames. In the second approach, we proposed a dynamic buffer scheme which is adaptive according to the current frame type. For the I-frame, the system will only request one frame buffer. When P- or B- image are to be decoded, 2 frames will be allocated. The effective number of frames per buffer is only 85% of the 3 frame buffer. The experimental results show that the buffer space is significantly reduced, and we observed a well-scaled decoding performance for the high resolution MPEG-2 video.

1.2 Multimedia Communication over WCDMA Wireless Network

Our investigation on parallelizing the MPEG-2 decoding algorithm indicates that real-time decompression of high quality video can be obtained via pure software solution. It is our belief that powerful computation power in the future will be available via massive integration of low-end CPUs into one chip/board. Thus our proposed data pipeline parallel decoder is particularly appealing for those systems equipped with multiple general purpose processors. Further more, our parallel MPEG-2 decoder also provides valuable suggestions to the design of a high performance parallel video encoder. With proper inter-node communication protocol and enough CPUs, software-based real-time encoding for high-quality video is possible.

However, the capability of real-time video encoding/decoding at the end-systems alone does not guarantee the quality along the whole path . We must also have a quality-guaranteed communication network to deliver the multimedia contents in a timely manner and with high fidelity. Multimedia data, especially streaming audio and video data, should be transported with delay guarantee in order to have a smooth playback. There had been considerable amount of research on this area. Some are trying to provide QoS over best effort network, such as RSVP [4] over INTERNET and VoIP[5]. There are also solutions with built-in QoS consideration at the network layer, such as ATM [6] and IPv6[7]. These research have shown that it is

becoming realistic to support a large number of multimedia connections with desired QoS over the wireline network.

Furthermore, distributed multimedia system should expand its coverage through a wireless communication link as well. The future generation wireless system needs to support a seamless integration (i.e., transparent application switching) to support voice, audio and conventional data (e.g., e-mails, and ftp). It should also support many users with guaranteed quality. However, the multimedia communication in the wireless network remains a technical challenge, due to the low information bandwidth and high transmission error in the physical layer.

The first generation wireless cellular network is designed to carry voice communication. These networks are analog system, and use frequency division multiple access (FDMA) to support multiple users. In FDMA system, the whole radio spectrum is divided into several isolated physical sub-channels, and each sub-channel is dedicated to a particular user once assigned by the base station. In order to reduce the cross-channel interference, sub-channels are often spaced apart by sufficient distance. Furthermore, neighboring cells are not allowed to use the same frequency band. However, these reduced the radio efficiency, and severely limit the number of concurrent users.

The second generation system is characterized by digital modulation and time division multiple access (TDMA) technology, which separates users in time. These systems start to provide limited support for data service, such as email and short message. However, TDMA is subjected to multi-path interference, where the received signal might come from several directions with different delay. Similar to the FDMA technology, neighboring cells in TDMA must use different frequency band to reduce inter-cell interference.

Nevertheless, the past decade have witnessed a rapid grows of the 2G system. Encouraged by the success of second generation cellular wireless network, researchers

are now pushing the 3G standard to support a seamless integration of multimedia data services, as well as voice service. However, multimedia data traffic is more demanding than voice service, both in data rate and maximum tolerated bit error rate (BER). For example, streaming video requires a low BER, less than 10^{-4}, and a moderate data rate (usually higher than 64 kbps). The bottom-line is that the minimum BER and data rate requirements for all admitted connections must be satisfied at any time. Otherwise the system will not guarantee the performance.

However, traffic channels in traditional TDMA based system have low data rate, and the bit error rate (BER) is often higher than required by multimedia traffic. Directly providing multimedia services based on the TDMA system, though possible, results in severe degradation in system capacity [8]. Thus, providing quality of guarantee service in wireless network needs new design and functionality in the MAC layer.

Another competing technology, code division multiple access (CDMA), uses a totally different paradigm to share radio resource among different users. In CDMA system, the whole spectrum is used as carrying band for all users in any time. Channelization is achieved by assigning different spreading codes to users. Each information bit will be spreaded into the baseband before transmission. The receiving side despreads the multiple occurrence of baseband signal back into the original bit. The interference caused by simultaneous transmission is reduced by the spreading/despreading process. For this purpose, the spreading codes are selected such that their cross-correlation is small.

CDMA technology is proven superior than FDMA and TDMA in many aspects [9]. For example, the multi-path signals in TDMA can be used to increase the overall signal quality by using multi-finger Rake receiver. Several advantages are listed here: First of all, the capacity in CDMA is much higher than FDMA and TDMA system, and the same frequency band can be reused by the neighboring cells. Secondly,

channels in CDMA are much secure, since the user's data are automatically encrypted during the spreading/de-spreading process. Thirdly, which is most important to multimedia traffic, is its capability of offering different level of BER and data rate by manipulate the spreading/despreading parameter, or dynamic spreading factor. Therefore, different types of traffic might co-exist in the system with minimal impact to each other. It is based on the above considerations, we choose WCDMA as the potential wireless solution to provide multimedia service and the direction for further research.

In this dissertation, we investigate effective protocol design with dynamic spreading factors such that various QoS based on different traffic types can be provided. Increasing spreading factors can benefit the system because it will increase the desired signal strength linearly. The measured bit error rate can be reduced 75 times with a long spreading factor. By taking advantage of this benefit, we propose some middle-ware solutions to monitor the network load and switch the spreading factors dynamically based on the current load with multimedia traffic. These middle-ware solutions are implemented in mobile and base stations, and experiments are performed to measure the actual system performance. The preliminary results indicated that our proposed system can always maintain a desired quality for all the voice connections. We further extended our protocol to guarantee a balanced support among different traffic types. While the voice communication is still guaranteed to be non-interrupted, the data traffic is proved to be served with reasonable response time by our proposed system.

We further extend our dynamic spreading admission control scheme to a multi-cell situation by proposing a dynamic spreading enabled soft handoff framework. The processing time of the handoff request was analyzed. We found that the update process caused by handoff is the major component of delay. Further investigation shows that the update associated with handoff might consume too much access channel

time and increase the delay of handoff, especially when handoff traffic is heavy. We, therefore, adopted a batch mechanism such that multiple handoff requests could be processed simultaneously. The average delay is reduced from 1.12 seconds to 800 ms in heavy handoff rate.

We also proposed a new Handoff Mobile Resource Allocation algorithm (HMRA) to optimize the performance of the mobiles in the handoff area. The spreading factor and transmission power for the handoff mobiles are jointly considered to maximize the throughput; meanwhile the algorithm maintains the BER requirements for the handoff mobiles and the target cell. The original problem is formulated in a nonlinear programming format. We proposed a procedure to simplify it into a linear constraint problem, which is solved by a revised simplex method. Numerical results show a 25% increase in throughput for WWW traffic, and a 26% improvement for the video traffic.

1.3 Dissertation Outline

The rest of this dissertation is organized as follows. Chapter 2 outlines related studies of high performance MPEG-2 video encoding and decoding. This research addressed different methods, including pure hardware based methods where specialized hardware architecture is used, hybrid methods which build multimedia operation into the general processor, and software parallel solutions.

In Chapter 3, we describe the framework of the proposed pure software based parallel MPEG-2 decoder. A data partition based parallel scheme is presented, in which the decoding of each video frame is subdivided to multiple slave nodes. The video frame is physically partitioned into several disjointed parts, and each part will be decoded at slave node. The performance results for the data partition algorithm with different partition methods are presented.

Chapter 4 presents the data pipeline based parallel algorithm. This method attempts to exploit the frame structure of the MPEG-2 bitstream. In order to reduce

the communication overhead and eliminate the inner-frame decoding dependency, we use a frame level parallelizing in the data pipeline scheme. The number of frames assigned to each slave node and the number of slave nodes become a design issue. Our analytical model and experimental results show that the data pipeline can provide close-to-linear speed up.

Chapter 5 and Chapter 6 constitute the second part of this thesis. In Chapter 5, we present a novel media access control (MAC) protocol for the WCDMA wireless network. The proposed protocol is designed to handle multimedia traffic. We demonstrate how the new protocol can guarantee the BER requirement. A new multimedia scheduling algorithm is described to utilize the dynamic spreading capability. Chapter 6 describes how the work can be extended to a multi-cell environment. A new handoff procedure is studied which is designed to coordinate the change of spreading factor when a mobile moves into a new cell. The handoff delay is analyzed for different traffic types. The analysis shows that a considerable delay could be caused by frequent handoffs. We thus use a batching process to reduce the number of unnecessary updates, which in turn reduces the handoff delay. The resource allocation for the handoff mobile is also discussed.

Chapter 7 summarizes the main contributions of this dissertation and presents some directions for my future research.

CHAPTER 2
MPEG-2 BACKGROUND AND RELATED STUDY

MPEG-2 [10] is one of the dominant digital video compression standards. Since its inception back in the early 90s, the standard is widely accepted and implemented by various hardware and software solutions. The standard defined the syntax of valid MPEG-2 bitstream, which covered a wide range of video quality (e.g., frame 352*240 QCIF video format to 1404*960 HDTV). Among various MPEG-2 applications, DVD (corresponding 720*480 resolution in the MPEG-2 family) is probably the most commercially successfully. However, we expect that high quality video format will become more desirable and need to be supported in the near future. For example, HDTV and even higher resolution video is being proposed for the next generation of electric consumer products. Therefore, a successful digital video solution should have good scalability performance. Specifically, the solution of interest should be able to be extended and provide satisfactory performance for the high-end high-quality video format. Our proposed software based parallel MPEG-2 decoding scheme is evaluated with a set of video streams, from low-quality, low bit-rate to high-end, high-resolution videos. As will be shown in Chapter 4, our investigation indicates that with some necessary revisions at the memory management algorithm, the data-pipeline scheme can produce a real-time decoding performance for high-end video streams.

In the MPEG-2 video encoding process, the raw video source is compressed by exploiting the spatial and temporal redundancy within the time-continuous video frames. Two key techniques used are DCT transformation and motion compensation. To reduce the spatial redundancy, the video frame is segmented into 8x8 pixel blocks, followed by the 8x8 2-D DCT transformation. The DCT coefficients are then

quantized to reduce the number of representing bits. The quantization step is a non-reversible process where part of pixel information is lost. Thus, the quantization table is designed to minimize the degradation of image visual quality. Studies in this field have shown that low frequency DCT coefficients should be assigned more bits than the high frequency part. Thus many of the high frequency elements will become zero after quantization. The quantized DCT coefficient matrix is serialized through a zig-zag scanning to concentrate the nonzero low-frequency DCT coefficients. Such a representation can be further compacted using a run-length encoding.

The MPEG-2 motion compensation technique further reduces the encoded bit rate. This is based on the observation that there is a lot of correlation between consecutive video frames. Using the concept of Group of Picture (GOP), video frames are encoded into three frame types: I-frame, P-frame and B-frame. Each GOP contains one I-frame, which is completely self-encoded without referring to other frames. The I-frame is presumely of the highest picture quality and serves as a reference frame for the P-frames and B-frames in the same GOP. The pixel blocks in the P-frames are encoded with forward motion prediction, where the reference frame is searched to provide a pixel block with highest match as prediction block, and only the residue is DCT-transformed and quantized. For the B-frame, two reference frames (I/P frame), one frame from the previous frame and another from the next frame, are used to perform a bi-direction motion prediction.

Thus the decompression should go through the following steps: First, the DCT coefficient and motion information are extracted from the run-length encoded bit-stream; this is followed by a de-quantization and an inverse DCT transform; then if the block is predicted from other block data, a motion compensation needs to be done to form the recovery image; then the dithering procedure will map the image into suitable color space of a particular display system.

High-performance software decoding for high-quality MPEG-2 video is becoming increasingly desirable for a wide range of multimedia applications. In the past few years, significant progress has been made in the microprocessor technology and software decoder optimization, brought real-time software-based MPEG-2 decoding [11, 12] into reality. Lee [12] implemented a real-time MPEG-1 decoder by fine-tuning Huffman decoding, IDCT algorithm and flattening the code to reduce the cost of procedure call/return. She also used shift and add operations to replace multiplication, which can be executed more efficiently with PA-RISC multimedia instructions. With the built-in multi-ALU and super-scalar features of PA-RISC, the cost of some decoding procedures can be reduced by up to 4 times. More dedicated multimedia instructions were introduced in many processor architectures and were used to accelerate MPEG-2 decoding process [13–15]. Bhargava et al. [13] conducted a complete evaluation of MMX technology for filtering, FFT, vector arithmetic and JPEG compression applications. Tung et al. [15] proposed an MMX-enhanced version of a software decoder by optimizing the IDCT and MC using Intel's MMX technology. They reported a real-time MP@ML decoding; however, the decoder was not fully compliant with the MPEG-2 standards. In Zhou et al. [16], a theoretical computation analysis of MPEG decoding was presented and the authors suggested using VIS (Visual Instruction Set) to achieve real-time decoding. However, their analysis only took into account the least arithmetic operations during the decompression stage, and the implementation results were yet to be reported. There are some public domain and commercial software decoders capable of real-time DVD (MP@ML) quality decoding [11, 16]. These products should not be cataloged as the pure-software solutions, since they are optimized with specific hardware multimedia support (e.g., Intel's MMX instruction set); thus they are still hybrid decoders. Another hardware-based approach takes advantages of a redundant DSP unit and the very wide internal

bus design, which make it possible to exploit instruction-level parallelism (such as VLIW); some of the work is reported in references [11, 17].

In summary, all of the solutions discussed above in deffering degrees depend on some specific hardware features in different degree, either from multimedia instructions in CPU or from a video display card. Thus these solutions are very difficult to transport to different hardware platforms. Furthermore, these solutions usually can not support scalability features of high profile MPEG-2 video. For example, in most of these schemes, the 30 frame per second (FPS) frame rate is used as the target performance. In some high-end profiles, however, a higher frame rate and resolution are required (e.g., MP@HL progressive video format defined a spatial resolution of 1920*1152 at a sampling rate of 60-fps [10]).

On the other hand, without the support of dedicated hardware, the computation of MPEG-2 decompression could be very demanding, especially for the high profile/level streams. The MP@ML MPEG-2 stream with one base-layer video stream needs at least 2-Mbps bit-rate (an average of 6-Mbps is used in DVD video). For some high-profile high-quality MPEG-2 videos, up to 40-Mbps could be necessary to support MPEG-2 extension layers (with MPEG-2 scalability features). The extension layers provide additional information that could be used in the decoding procedure to produce more vivid video quality. They also require additional computation in the decompression procedure. The decoding of the extension layer has to go through a procedure similar to the base layer; thus the decoding time of a scalable MPEG-2 video stream will increase proportionally to the number of layers. For a stream with two scalability features (temporal scalability and SNR scalability) at the spatial resolution 1920*1152 (the image size is six times higher than that of MP@ML quality video), the required computation is roughly $3 * 6 = 18$ times that of MP@ML MPEG-2 video decoding. This huge amount of computation requirement makes it very difficult for the real-time playback of a software-only decoder, even with

the fastest CPU available. Hence, functionality-based or data-based parallelization should be considered to boost the decoding performance.

Various parallel schemes have been studied for video encoding in the literature [18–20]. In Akramullah at al. [19], a data-parallel MPEG-2 encoder is implemented on an Intel Paragon platform, reporting a real-time MPEG-2 encoding. Gong and Rowe [20] proposed a coarse-grained parallel version of a MPEG-1 encoder. A close-to-linear speed-up was observed. Comparable work of a parallel MPEG-4 encoder has been reported [21]. The authors used different algorithms to schedule the encoding of multiple video streams over a cluster of workstations. A parallel MPEG-2 decoder based on shared-memory SMP machine was reported [22]; however, they did not address how real-time decoding should be supported for the high-profile and high-level video source. In Ahmad et al. [18], the performance of the parallel MPEG encoder based on the Intel's Paragon system was evaluated. In their work, the *I/O node* provides several raw video frames to the *compute nodes* in a round fashion, and each *compute node* performs the MPEG encoding, sends back the encoded bitstream, and requests more video frames from *I/O node*. The *I/O node* serves the requests from *compute node* in a FCFS manner, which may reduce the idle time in the *I/O node*. However, this scheme allows out-of-order arrival of encoded frame data and the *I/O node* has to track the dynamics of the assigned frames for each compute node. This increases the complexity in the *I/O node* and requires much higher buffer space to preserve the original sequence of video frames. In our case, since the master needs to display the decoded video in exact order, it must buffer all the decoded frames before the arrival of precedent video frames. The build-up of frame buffer might exceed the physical memory in master side if the same scheduling is used. Thus we use an in-order data distribution between master node and slave node.

Yung and Leung [23] implemented an H.261 encoder on an IBM SP2 system. Both spatial and temporal parallelization were investigated. Their results showed

MB level parallel could equally provide a high speed-up as the frame/GOP level parallelization. However, this claim was not true in the parallel decoding. Our results show that a data-partition scheme (MB level) can only provide limited performance gain when the number of slave nodes increases. Furthermore, the degree of scalability highly depends on the communication bandwidth, which was not fully exploited in their study. By carefully investigating the impact of a Master/Slave communication pattern, we found that the communication pattern in parallel decoding can be more complicated than in the case of encoding, which eventually becomes the key factor determining the peak system performance.

CHAPTER 3
DATA-PARTITION PARALLEL SCHEME

Our initial investigation focused on the data-partition scheme. We consider data partition on the macroblock level since the majority of MPEG-2 decompression computation is spent on block level IDCT and motion compensation. This scheme allows a low complexity in each computing node, since only part of the MPEG-2 decoding procedure needs to be implemented in the computing node. The data-partition scheme also has the advantage of the potential quick response to the end-user commands. As explained in the rest of this chapter, a video frame is divided into several groups of macroblocks that are decompressed in slave nodes simultaneously. Thus the system can response very quickly (always less than the decoding time of one frame for non-preemptive scheduling). In fact, the response time is inversely proportional to the parallel gain in the data-partition scheme. An important design factor in the data partition algorithm is the parallel (partition) granularity. It is possible to divide a frame in pixel-level; however, such a fine grain scheme may introduce too much overhead. We believe that the macroblock level data partition scheme should be a natural choice for our initial investigation. In the following discussion, all partition algorithms are based on macroblock level data decomposition. To maximize the decoding performance, the following issues should be considered:

- In general, the percentage of the parallelized computing components determines the upper bounder of potential performance, which indicates that we need to parallelize as many decompression steps as possible. Our preliminary experiments show that the processing of macroblocks takes about 95% of the total computation. Thus by adopting a macroblock level parallelizing, we expect that majority of the computation is parallelized.
- New overhead caused by the parallel scheme should be analyzed. Specifically, due to the inter-frame and intra-frame dependence, our data-partition scheme

will introduce additional communication cost. For example, motion compensation may require reference pixel blocks of previous frame. How to reduce the data communication cost becomes critical. As we will see later, different partition algorithms can result in different amounts of additional reference data. Finding the optimal partition to minimize the communication overhead becomes a main challenge.

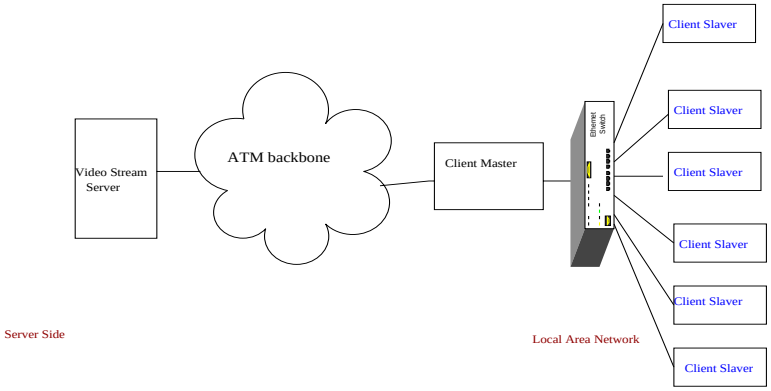

Figure 3.1: Parallel decoding architecture

Figure 3.1 depicts the system architecture of our parallel decoder. The system consists of a master node, which receives the MPEG-2 encoded video stream from a remote video server, and several slave nodes to do the decompression. To distribute the whole decompression workload to the slave nodes, the master node splits the bitstream into frames. Each frame, as a set of macroblocks, is further divided into N parts, for the N slave nodes. These N sets of raw data, together with necessary reference picture data, will be sent to the slave nodes for decompression. The master node has to wait until all slave nodes finish their decoding job, and send the decoded macroblocks back to the master node. The decoded macroblocks are then merged into one complete image for display. The majority of the MPEG-2 decompression steps (e.g., IDCT, MC, Dithering) are performed at slave nodes. The master node has its main duty in bitstream parsing and subtasks distribution for slave nodes. The above procedures are illustrated by the following pseudo-codes:

(1) Control algorithm in the master node. The master node receives the bitstream from the server, extracts frame data, partitions it into subtask, packed with

reference frame data, and sends it to slave nodes. Then we wait for the decompressed data.

```
Procedure Master_Control
  Initialize inside buffer and start slave nodes
  WHILE (there is more data in input bit-stream
         from server)
       Rbuff = Buffer for the current frame data
       Perform a partition over Rbuff, result with a set of N
            sub-task buffers:
               P1, P2,...PN /* refer to the
                  partition algorithms in next section */
       Update new reference data for each slave Ref_i
       FOR(i =1 to N)
            Send to slave i with P_i and reference data R_i
       END FOR
       Wait to Receive decoded micro-blocks from each slave node
       Combine the parts frame into the whole picture
       IF (current frame is on the top of display buffer)
            call display routine
        END IF
  END WHILE
End Procedure
```

(2) System Level Algorithm of Slave Node for Data Partition Scheme: Slave nodes receive the bit-stream from server, perform MPEG-2 decoding procedure, and send back the decompressed micro-blocks to the Master node.

```
Procedure  Slave_Decode
  Initialize inside buffer and
  set up session decoding parameter(from master node)
  WHILE (no terminate signal from Master node)
         Receive raw data from master
/*perform MPEG-2 decoding over the given portion data,
including:*/
         Inverse Run-length decoding to restore DCT
                coefficient/motion vector
         Inverse DCT
         Perform motion compensation
         Perform dithering
         Send to master the decoded micro-blocks
  END WHILE
End Procedure
```

3.1 Performance Model of Data-Partition Scheme

The performance of our data-partition scheme can be represented by the frame decompression time $T_f(P)$, with respect to a given partition P. The determination of $T_f(P)$ depends on (1) the computation in slave nodes T_c, (2) the transmission time for the interprocessor communication T_t, and (3) the housekeeping computation in the master node T_m. Since the decompression in slave nodes and communication can be performed simultaneously, T_f should be dominated by the longer computation path. We have

$$T_f = T_m + max\{T_c, T_t\} \tag{3.1}$$

Table 3.1: Notions used for decoding modeling with data-partition scheme

H	number of macroblocks of a frame in vertical dimension
W	number of macroblocks of a frame in horizontal dimension
$F = \{m_{i,j} : 0 < i < H$, $0 < j < W\}$	the set of macroblocks for a frame
K	number of total slave nodes
P	A particular partition of F into K parts (subsets)
$T_f(P)$	decoding time of one frame with respect to partition P
TC_P	communication time per frame for a given partition P
$S_{P,k}$	Subset of F assigned to k_{th} slave node
T_k	decoding time of $S_{P,k}$ at k_{th} slave node
$C_P(k)$	The amount of data communication per frame at k^{th} slave node
B	Network bandwidth

Statistics show that T_m could vary from 5% to 10% of the sequential decoding time and we use the higher boundary for the sake of brevity. Table 3.1 listed the terminology used to calculate the other two costs. Partition P consists of K disjoin subset of macroblocks $S_{P,1}\, S_{P,2} \cdots S_{P,K}$, where $\bigcup_{i=1tok} S_{P,i} = F$. The computation is performed simultaneously on slave nodes. For each slave node, a different amount of decompression time is required based on the decoding workload (size of each $S_{P,k}$)

and CPU power (i.e., CPU clock rate). Therefore, from the view of the master node, the slowest slave node determined the overall decompression time.

The communication costs between master and slave nodes should include the transmission of raw data and decoded data. Aside from raw data, the master node must send reference data to slave nodes, because of the motion prediction technique used in MPEG-2.[1] Given a frame partition P, the frame decompression time can thus be expressed as

$$T_f(P) = max\{max_k\left(|S_{P,k}| \cdot T_k\right) + \frac{\sum_k C_P(k)}{B}\} \tag{3.2}$$

Thus the decompression time is a joint effect of system hardware parameter (T_K, B etc.) and the partition P(both $S_{P,k}$ and $C_P(k)$ are functions of P). Given a hardware environment, an optimal partition should be found to minimize the decompression time. In another words, the optimal partition should minimize T_f over all feasible partitions.

$$P^* = argmin_{P\in\{allfeasiblepartitions\}}T_f(P) \tag{3.3}$$

It can be shown that, to minimize the first term in the right side of equation (3.2), we should assign the subtasks $S_{P,k}$ in proportion to the processing power of k^{th} slave node. For example, in a two-slave-node situation where $T_1 : T_2 = 1 : 4$, the optimal partitions should have the property of $|S_1| : |S_2| = 1 : 4$, where $max_k\left(|S_k| \cdot T_k\right)$ is minimized. However partitions with such property might not result in a minimum for the second term in (3.2). For example, the second term will always be minimum when all the macroblocks are assigned to one slave node. This partition unfortunately results in the longest overall computation time. The trade-off between computation and communication makes it a nontrivial work to find the optimal (slave number, partition) pair. Before we further analyze equation 3.2, we should establish a direct relation between the data communication $C_P(k)$ and the partition P.

[1] Macroblocks of P and B frame may be encoded using motion prediction. They are predicted by macroblocks from the searching area of a previous I or P frame. The search area is defined as a rectangle that centered in the predicted macroblock.

3.2 Communication Analysis in Data Partition Parallel Scheme

For a given partition P, the communication between the k^{th} slave node and master $C_P(k)$ can be further divided into three parts: (1) the amount of raw data from the master node to the k^{th} slave node, which is approximately $\|S_{P,k}\| \cdot s_m$ (assuming that a macroblock in MPEG-2 bitstream requires an average of s_m bits), (2) because of the motion compensation, MPEG-2 has a strong inter-frame data dependency. The encoding of P frame and B frame requires other (previous) I or P frames as references. This is the very reason that causes additional inter-nodes data communication. We use $R_P(k)$ to represent the reference data at the k^{th} slave node under partition P. (3) the decoded macroblocks in the k^{th} slave node requires $\|S_{P,k}\| \cdot 16^2 \cdot 8$ bits, assuming one byte per pixel in the 16 by 16 macroblock.

Therefore the communication time (second term of equation (3.2)) can be rewritten as

$$TC_P = \sum_{k \in \{1 \cdots K\}} \frac{(\|S_{P,k}\| \cdot s_m) + (\|S_{P,k}\| \cdot 16^2 \cdot 8) + R_P(k)}{B} \tag{3.4}$$

It is noticed that $\sum_{k \in \{1 \cdots K\}} \|S_{P,k}\| \cdot s_m$ is exactly the size of one frame in MPEG-2, and $\sum(\|S_{P,k}\| \cdot 16^2 \cdot 8)$ is the size of a decoded frame. We have

$$TC_P = \frac{S_e + S_d + \sum_{k \in \{1 \cdots K\}} R_P(k)}{B} \tag{3.5}$$

here S_e and S_d represent the compressed frame size and the original picture size respectively. Given an MPEG-2 video stream, the average S_e is obtained from the encoded bit-rate, and S_d comes from the image horizontal size and vertical size encoded in the bitstream. The total reference area in the k^{th} slave node $R_P(k)$ is the union of the reference area for each macroblock in $S_P(k)$. According to MPEG-2, the predicting macroblock is selected from a searching window, which is defined as a square area centered in the target macroblock. The dimension of the searching area is usually fixed through a given video title. Let s be the searching window size predefined in encoding time, the reference area for one macroblock is $(16+s)*(16+s) = 16^2+32*s+s^2$

pixels. The difference between $R_P(k)$ of current frame and $S_P(k)$ of the refereed frame has to be transmitted from master to slave node.

To simplify our discussion, we assume a static partition scheme where the partition is determined at the initial phase and fixed through the decoding session. The fixed partition in fact provides a chance to reduce the $R_P(k)$: since the same area of each frames is decompressed in the same slave node, the difference between $R_P(k)$ and $S_P(k)$ can be reduced significantly. An upper bound of $R_P(k)$ is given by

$$R_P(K) \leq \|S_P(k)\| \cdot (s^2 + 32 \cdot s) \tag{3.6}$$

With static partition, the reference area only needs to be transferred once per GOP (Group of Picture), since it can be reused by the successive B-frame (not separated by I- or P-frame). Therefore the transmitted reference data can be further reduced. Assume that the system GOP pattern is $I : P : B = 1 : b : c$, there will be a total of $(1 + b + c)$ frames in one GOP. The referred frames can only be I and P frame. Thus the required reference transmission ratio is $\gamma = \frac{1+b}{1+b+c}$. The amount of reference data should be averaged over a GOP. TC_P is revised as

$$TC_P = \frac{S_e + S_d + \gamma \cdot \sum_{k\in\{1\cdots K\}} R_P(k)}{B} \tag{3.7}$$

Substituting equation (3.6) into (3.7), the total communication should be bounded by

$$\begin{aligned} TC_P &\leq \frac{S_e + S_d + \gamma \cdot (s^2 + 32 \cdot s) \sum_{k\in\{1\cdots K\}} \|S_P(k)\|}{B} \\ &\leq \frac{S_e + S_d + \gamma \cdot (s^2 + 32 \cdot s) \cdot H \cdot V}{B} \end{aligned}$$

3.3 Partition Example and Communication Calculation

Our first partition example ($P1$) is a result of the Horizontal-Vertical(HV) Partition algorithm, which is defined by the following recursive procedure: if the previous partition operation is performed along horizontal dimension, then we partition

the image vertically for this time. If the partition number K is even, then we separate the working area into two parts with equal size, with the partition dimension decided above. Otherwise, the whole area is partitioned into two pieces with area ratio of $1 : (K-1)$. For each of the two resultant parts, we perform the above steps correspondingly.

By applying the $HV - Partition$ algorithm with input $K = 4$ (i.e., 4-way partition), we have our first partition $P1$, shown in Figure 3.2. $P1$ separates the original picture into 4 adjuncted areas: A, B ,C *and* D. We found that the reference area $R_{P1}(k)$ consists of the belt-shape-area along the internal edges. The width of the belt is the searching window size s. A simple way to calculate the total reference area is to add the length of all internal edges.

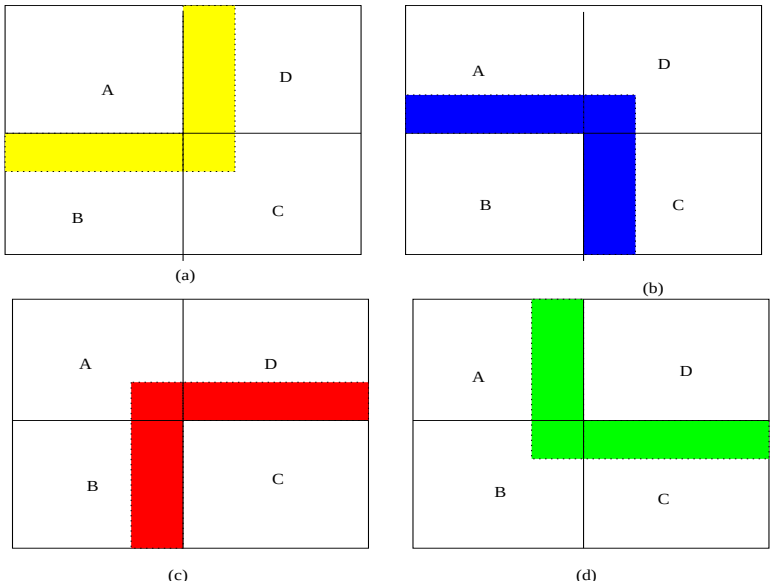

Figure 3.2: Horizontal-vertical partition into four piece A B C and D.

For a 720*480 picture, each part contains two internal edges , resulting in a total length of $360 + 240 = 600$ pixels. The total reference area for this partition is thus $4 * 600 = 2400$ pixels. Each pixel contains 3 color elements(each require 8 bits). With a searching window size of 32, the total amount of reference data is: $\sum R_{P1}(k) = 4 \cdot (600 \cdot s \cdot 8 \cdot 3) = 1.8 MBits$. For a GOP structure of IBBPBBPBB,

$\gamma = \frac{1}{3}$. The total communication time under 80Mbps (sustained) network bandwidth should be $C_{Pvh} = \frac{S_e + S_d + \gamma \cdot \sum_{k \in \{1 \cdots K\}} R_P(k)}{B} = \frac{0.5 + 2.76 + 0.6}{80} = 48\ msec$

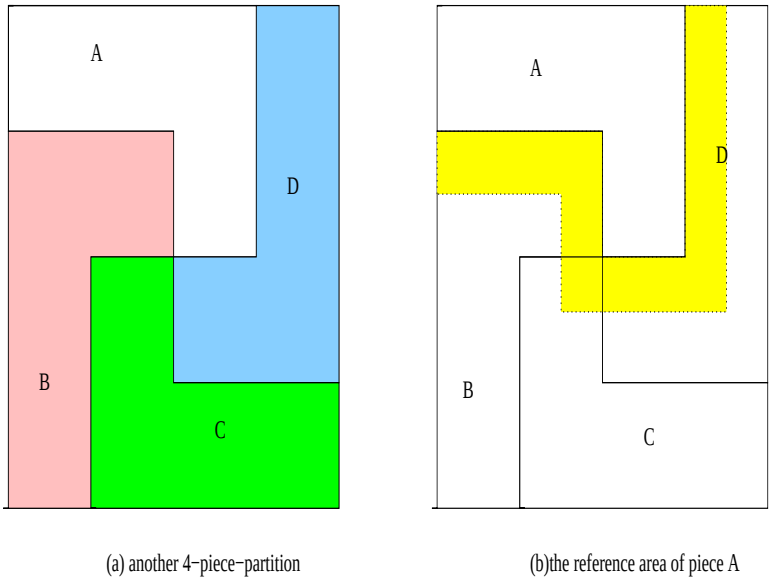

Figure 3.3: Another way to partition a picture into 4 parts

An alternate partition is shown in Figure 3.3.(a). There are 8 internal edge segments, separating the picture into four identical pieces. The reference area of the upper-left piece is shown as gray area in Figure 3.3.(b). Using the same calculation method as in the above example, the total reference data increases to $3600 * 32 * 8 * 3 =$ 2.8 M-bits. Compared to the previous partition, this partition require 50 % more reference data. The total communication cost increases to 4.19-Mbits/frame.

3.4 Communication Minimization

To obtain the minimum communication cost, the best partition should be found, given the image shape and the number of slave node. One approach is to use brute-force searching. However, the size of searching space is prohibitively large even we enforce partitions to be equal-size. The number of possible partitions can be figured out as following: Let $H * V$ be the number of macroblocks for a frame and there are K slave nodes. The problem is the same as putting $H * V$ objects into K boxes. This is equivalent to the combination of choosing $X = \frac{H*V}{K}$ objects

from $H * V$ objects, followed by choosing another X from what is left, until no more selection. The total number of the combinations is

$$\begin{aligned} & C_X^{H*V} \cdot C_X^{H*V-X} \cdots C_X^{2*X} \cdot C_X^X \\ &= \frac{(H*V)!}{X!} \cdot \frac{(H*V-X)!}{X!} \cdots \frac{(2*X)!}{X!} \cdot \frac{X!}{X!} \\ &= H*V \frac{(H*V)!}{(\frac{H*V}{K})!^K} \end{aligned}$$

For a 1K by 1K picture, there are 4096 macroblocks. When partitioned into four slave nodes, the number of partition is in the order of 10^{690}. For $K = 8$, it is about 10^{1922}. For $K = 16$, this number easily exceeds 10^{3155}. The size of the searching space increases exponentially with picture size. To avoid this explosive searching space, some heuristic algorithms are desired.

The partition problem can also be addressed as an instance of quadric assignment problem–a classical combinatorial optimal problem, which has proven NP-hard. Using the notation in Table 3.1, the equal-size partition implies: For the K subsets $S_{P,k}$ of frame F, the following properties hold: $|S_{P,i}| = |S_{P,j}|$, $\bigcap S_{P,i} = \phi$ and $\bigcup S_{P,i} = F$, for $0 < i, j < N$. Under this assumption, the optimization of T_f is equivalent to the optimization of T_t (see Equation (3.2)). Let the total N (N=H*V) macroblocks be indexed from 1 through N. The object function is:

$$min(\sum_{i,j \in \{1 \cdots N\}} \sum_{k,h \in \{1 \cdots K\}} X_{i,k}.X_{j,h}.d_{k,h}.r_{i,j}) \tag{3.8}$$

The problem becomes the determination of N*K variables $X_{i,j}$, $1 \leq i \leq N$ and $1 \leq j \leq K$, that minimizing equation (3.8). $X_{i,j}$ is subject to the following constraints:

$$X_{i,j} \in \{0, 1\} \tag{3.9}$$

$$\sum_i X_{i,j} = \frac{H \cdot V}{K} \tag{3.10}$$

$$\sum_j X_{i,j} = 1 \tag{3.11}$$

Here $X_{i,j} = 1$ when macroblock i is assigned to the j^{th} slave node. Equation (3.11) shows that each slave node is assigned $\frac{H \cdot V}{K}$ macroblocks, and each macroblock can be assigned to only one slave node. $d_{k,h}$ is the transmission time to send an unit of decoded data from the h^{th} slave node to the k^{th} slave node, or vice versa. We assume that the communication bandwidth between any pair of nodes are identical (i.e., a homogenous environment). $r_{i,j}$ represents how much data in the i^{th} macroblock happens to be the reference data for the j^{th} macroblock. This is determined by the relative position of the two macroblocks in the image. It is obvious that $r_{i,j}$ is symmetric, and is in fact the intersection between s_i (searching window area of macroblock m_i) and m_j. When $s \leq 16$, $r_{i,j}$ is defined by

$$r_{i,j} = \begin{cases} 0 & i = j \\ 16 \cdot s & m_i \textit{ and } m_j \textit{ are neibour} \\ s^2 & m_i \textit{ and } m_j \text{ intersect at conner} \\ 0 & \textit{otherwise} \end{cases}$$

3.5 Heuristic Data Partition Algorithm

For general QAP problems, genetic algorithms and simulated annealing algorithms (SA) have be studied intensively [24, 25]. Unfortunately, when applying to our problem, these methods fail to produce satisfactory performance. The reference data obtained from SA algorithm with input sizes of 100, 300 and 500 macroblocks are 8448, 35712 and 56448 pixels respectively. The SA algorithm takes 1 minute, 30 minutes and 5 hours respectively. When using the HV partition algorithm, for the same input size, the costs are 7680, 15360 and 17280 pixels respectively, and the running time is in the level of several seconds. The HV algorithm takes advantage from the problem nature by grouping macroblocks in its natural layout. Further more, we proposed a more dynamic partition algorithm that exploits more information from the shape of partitioned area, named Quick-Partition:

1. Given input (K, hd, wd, S), where K is the number of current partition, S is a rectangle area to be partitioned, hd is the length of S in horizontal dimension, and wd is the length of S in vertical dimension.

2. If K is less than 2, than no further partition is required.

3. Calculate $k_1 = \lfloor \frac{K}{2} \rfloor$ and $k_2 = \lceil \frac{K}{2} \rceil$ as the area of two small rectangles to be generated from S.

4. If $hd \geq wd$, we find a vertical line that separates the rectangle S into two sub-rectangles with area ratio of $k_1 : k_2$. Otherwise, a horizontal line should be found similarly.

5. Calculate the width and height for the two sub-rectangles. Name it as S_1 and S_2.

6. For each of S_1 and S_2, we perform the same procedure from step 1 to 6.

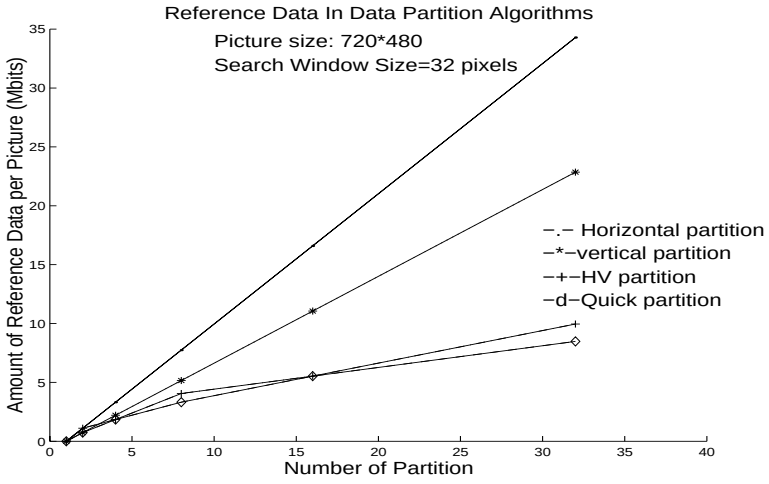

Figure 3.4: Performance comparisons of different partition algorithms

Figure 3.4 demonstrated the amount of reference data as a function of partition number K and different partition algorithms. We compared the performance of four algorithms: Horizontal-Partition, Vertical-Partition, HV-Partition and our

Quick-Partition algorithm. The Horizontal-Partition always divides the frame in horizontal dimension, while Vertical-Partition does the partition using vertical line. HV-Algorithm partitions the frame with interleaved horizontal line and vertical line. These algorithms are tested over 720*480 picture, with the searching window size set to 32. For Horizontal-Partition algorithm, the reference data increases linearly as the number of partition K increases. A similar (linear increasing) $R(K)$ trend is observed for the Vertical-Partition algorithm. The slope of the curve, however, is less than that of Horizontal-Partition. This is because the width of a video frame is actually larger than the height. At $K = 4$, the Vertical-Partition algorithm results in an about 2.8 M-bits/frame reference data, which is 33 % less than that of the Horizontal partition. The same percentage of reduction holds for other values of K. The HV-Partition algorithm generates a much-better result in terms of reducing the reference data. For partition number $K = 4, 8, 16$ and 32, the corresponding $R(K)$ are 1.8432, 4.055, 5.5296, and 9.9533 M-bits/frame respectively. Compared with the Horizontal or the Vertical partition, there is remarkable reduction in $R(K)$, especially for high partition number. For example, the reductions of the reference data (compared with the Vertical-Partition) at the above points are 16.67%, 21.43%, 50% and 56.45 % respectively. Our Quick-Partition algorithm produces the least amount of reference data in the four algorithms. In general, the Quick Partition performs either equally or better than the HV algorithm. The resultant reference data of the Quick partition for K =2, 8 and 32 are 33.33% , 18.18%, and 14.81% less than that of HV-Partitions respectively.

3.6 Experimental Results

The performance of the data-partition scheme is verified by experiments over two hardware/software configurations:

- 100-Mbps LAN: A 100Mbps Ethernet, and each node is equipped with Pentium-II 450-MHz CPU and 128-MB memory. The operating system is RedHat Linux6.2.

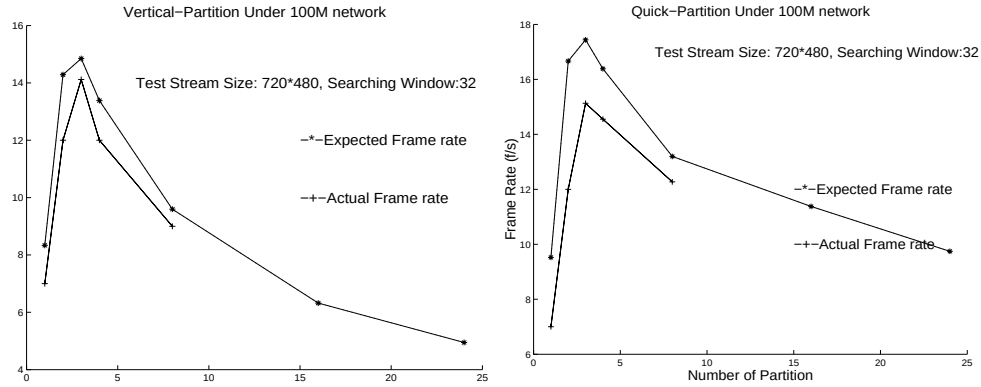

Figure 3.5: Decoding rate under 100Mbps network

- 680Mbps SMP: This environment is actually a symmetric multi-processors server machine. Each processor is a 248-MHz Ultra SPARC processor. The actually inter-node communication bandwidth can reach up to 680-Mbps, based on our measurements. This machine is used to emulate a Gigbits local network.

3.6.1 Performance over A 100-Mbps Network

The 100Mbps network has a total of 16 slave nodes equipped with Pentium-450 MHz processor. Each node is capable of decoding at 10-fps (with compiling optimization). This configuration is widely available in many organizations or companies. The performance of our partition algorithm is depicted by Figure 3.5. Figure 3.5.a shows the performance when the Vertical-Partition algorithm is used, and figure 3.5.b is of the case of the Quick-Partition algorithm. The following observations can be made:

- For small value of K, both partition algorithms produce close-linear performance improvements. For the Vertical-Partition algorithm, the predicted decompression rates for K=1 and 2 are 8-fps and 14-fps respectively. The actual frame rates are 7 and 12-fps, which is very close to our expectation.
- The Quick-Partition algorithm has a similar result as that of the Vertical-Partition when K is small ($K <= 3$). The predicted and observed decompression rates are nearly identical. This is because when the number of partition is small, the slave decompression time still dominates the decoding cost. The difference in reference data of the two partition algorithms is overwhelmed by the decompression time at slave nodes.
- At K=3, the cost of communication begins to play a determinant role. The benefit of the reduced node-decoding time is canceled by the increasing of communication time (which is mainly caused by the reference data). The decoding rate at K=3 is only 14-fps, which is only 16 % higher than that of $K = 2$.

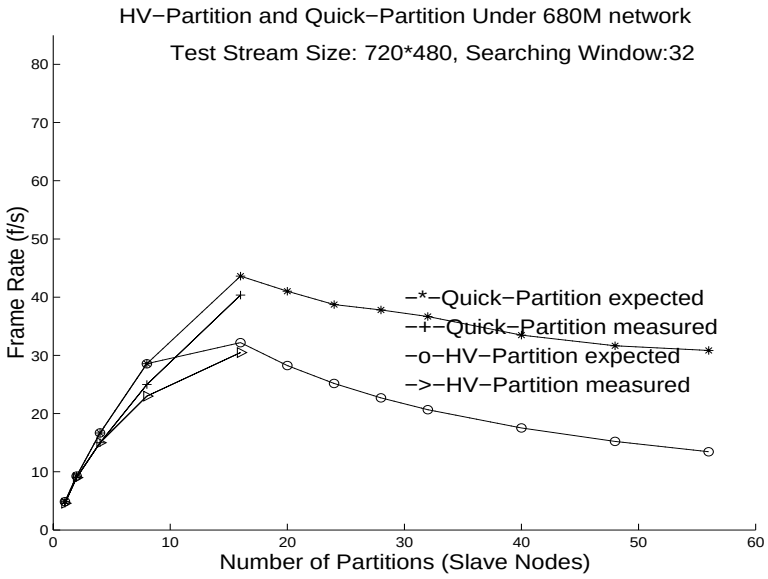

Figure 3.6: Decoding rate under 680Mbps environment

- $K = 3$ represents the peak decompression rate of this configuration. Further increasing K results in a degrading of the performance. For the Vertical-Partition algorithm, we observed 12, 9, and 6-fps for K=4, 8, 16 respectively. The performance degrading of the Quick-Partition algorithm is not so severe. The decompression frame rates are 16, 13 and 11-fps respectively. The performance improvement is limited, since the network is not fast enough.

3.6.2 Performance over a 680-Mbps Environment

This experiment is performed with a 15-node SMP server. Each node contains an Ultra SPARC 248-MHz CPU. When used along, each CPU can decode at 5-fps . The inter-node communication bandwidth is up-to 680-Mbps. This environment is used to emulate a Gigbits network. The performance results for the HV-Partition and the Quick-Partition algorithm are presented in Figure 3.6.

It is observed that the overall performance of our data partition parallel scheme improved significantly even though the processing power at the individual node decreased. Detailed discussions are listed below:

- When using HV-Partition, we observe a close-linear increasing of the decompression rate until $K = 8$. The predicted frame rate are 4.8, 9.3, 16.7 and 29-fps for K=1, 2 ,4 and 8 respectively. The actual frame rates are only slightly less than the predicted values (the difference is less than 10 %). When K is in the range of 8 to 14, the decompression rates keep increasing, however the speed of increasing is slowed down. For example, when the number of slave node doubles

from 7 to 14, the decompression rate (actual) increases from 23-fps to 30-fps (only 30 % performance gain).

- For the Quick-Partition algorithm, we observed considerable performance improvement. The close-to-linear speedup is sustained until $K = 16$, where the theoretical peak decompression rate of 44-fps is expected. The actual decoding rate match with our model closely, the decoding rates for K=1, 2, 4, 8 and 14 are 4.6, 9.0, 15, 25 and 40.4 respectively.

- At K=16, the decoding performance reaches its peak for the Quick-Partition and Vertical-Partition algorithm, and the overall system performance start going down. Again, this is because the communication cost of reference data overwhelmed the capability of system communication. However, due to the lack of more processing nodes in our SMP server, we are unable to verify the performance trend after K=14.

CHAPTER 4
PIPELINE PARALLEL SCHEME

The major overhead of the data-partition scheme exists in the master node, where the compressed video stream needs to be parsed down-to the macroblock level. This portion of computation becomes significant when the number of slave nodes increases, thus prevents further improving of decoding rate. Unfortunately, multi-master nodes (as used by N. H. C. Yung et al. [23]) will not work since the bitstream is VLC encoded and highly auto-correlated. An alternate scheme for further performance enhancement relies on the increasing of decoding unit, by distributing a block of frames to each workstation.

4.1 Design Issues of Pipeline Scheme

Several design issues should be addressed in order to have a high parallel processing gain.

- The idle time at the slave nodes should be minimized. There might be a waiting time if a slave node can not receive new block of frames from the master node, once the decompression for the previous frame is finished. For this purpose, the master node is designed such that it filled new data to other slave nodes during the computation time of a particular slave node. Therefore, the waiting time in the slave nodes is minimized. Coupled with this concern is the load balancing between slave nodes. A successful parallel scheme should avoid overwhelming of some nodes while starving some other nodes. Our experiment results show that the STD of CPU usage of the slave nodes is low, and reduced to less than 0.03 when block size is set to one GOP. We believe this fact will hold regardless the video contents being tested, as far as the underline slave nodes are homogeneous. A more adaptive task scheduling should be favored for the case of heterogeneous environment. It seems that a static assignment of block size based on the CPU speed of the individual slave node should be sufficient. We will not discuss this dimension for the sake of the space.
- The workload in the master node should be minimized. In fact, the master node represents the only sequential link in the pipeline scheme. The master node must perform MPEG-2 header parsing so that the frame data can be extracted for each slave node. By developing a quick bitstream parser which

scan only the GOP start code and picture start code, the computation of this procedure is negligible compared to the other MPEG-2 decoding steps.

- An important design factor is the proper block size for each subtask. Should the master node deliver 2 frames of raw data to a slave node each time? How about 5 frames or 10 frames? We have found that a proper block size does influence overall system performance significantly. In fact, the overall system communication is mainly determined by the block size, due to the inter-frame data dependency and the I-P-B frame structure in the encoded bit-stream. For example, a slave node that decodes a P-frame will need an I-frame as the reference. If the block size is not chosen carefully, the referred I-frame can be in another node, thus we need to transmit one full decoded I-frame to the target node. Notice that this also introduces additional delay which can significantly affect the pipeline efficiency. For the worst case, a previously decoded I-frame may be required by all slave nodes, if the following P- or B-frames happen to be distributed all over these slave nodes.
- Another important design issue is the determination of the optimal number of slave nodes. This is particularly important in a practical environment. The question can be put as following: given a certain hardware configuration, how many nodes will the system need in order to reach the peak performance? We have found that the network bandwidth is one physical limit. Given a certain network, we are able to predict and verify the maximum number of the slave nodes that will deliver the best decoding rate. Further increases on the number of the slave nodes will not generate a positive effect to the decoding rate.

In the following discussion, we present the pseudo code of our proposed parallel algorithms. The detailed analysis on the communication overhead and performance prediction will be addressed later. For simplicity, a homogeneous environment is assumed where slave nodes have identical computing capability.

Algorithm 1 (Master Node): For each of the slave nodes, The Master node extracts a block of frames from the incoming bit-streams, and delivers it to the slave nodes. This procedure repeats until the end of a video session. D is the block size, N is the number of slave nodes.

```
Procedure Pipeline_Master(D)
        Initialize internal buffers and start slave nodes.
        rnd = 1 /* set rnd to 1 for the first round*/
        FOR (j=1 to N)
               /* Pipeline initialize, fill the slave nodes with
             raw data to start the pipeline. */
                send a block of D frames to the jth
```

```
                slave node for decoding
        END FOR

        WHILE (There is data in the incoming buffer)
            FOR (j =1 to N)
                Receive Decompressed data:
                        (rnd − 1).N.D + (j − 1).D^th frame to
                        (rnd − 1).N.D + j.D^th frame
                from j^th slave
                Prepare a block of raw data:
                        Extract rnd.N.D + (j − 1).D^th frame
                        to rnd.N.D + j.D^th frame raw data
                from the incoming buffer
                R = the reference I frame needed
                send F and R to j^th slave
            END FOR
        END WHILE
END Procedure
```

Algorithm 1 depicts the control flow on the master side. Before entering into a stable pipeline running, the system has a start-up procedure to establish the pipeline. The master node then enters into round operation represented by the for loop block. The interaction between the Master node and the Slave nodes at each round is synchronized to have a continuous pipeline. During each normal round, the Master node exchanges data with the Slave nodes in a round-robin manner. For each of the N Slave nodes, The master node will first receive D frames of previous round that are decompressed at that Slave node. It will then send a block of D compressed frames to the slave node. When it moves to serve the next one, the slave node will receive new data and do the decompression simultaneously.

Algorithm 2 (Slave Node) : Slave nodes receive a block of D frames each time from the Master node, and decompress it

```
Procedure Pipeline_Slave()
        Initialize internal buffers, receive decoding
          parameter and D,N values from Master
        WHILE (no terminate signal from Master node)
```

```
                Receive D frames of raw data from master
                /* perform MPEG-2 decoding for the given frames
                data,including:*/
                  FOR (each of the D frames)
                        Run-length decoding to restore
                            DCT and motion vector
                         perform Inverse DCT
                        perform motion compensation
                        perform dithering
                END FOR
                send to master the decoded frames
        END WHILE
END Procedure
```

As mentioned early, it is possible that the required reference frames (i.e., I- and/or P- frames) are not available locally. Thus, these extra reference frames needed to be transmitted to the proper nodes, in addition to the raw video block. How much is the required extra transmission of these reference frames? What are the optimal values of D and N such that these extra transmissions can be minimized? The following example demonstrates the need for a proper D value. We assume that the video shot is encoded with a fixed GOP (Group of Picture) length L and fixed frame pattern within a GOP (e.g., I:P:B=1:2:12). The following sequence depicts the GOP pattern $G = \{I_1, B_1, B_2, B_3, B_4, P_1, B_5, B_6, B_7, B_8, P_2, B_9, B_{10}, B_{11}, B_{12}\}$.

- If $D = 1$, with two Slave nodes ($N = 2$), the first Slave node has a frame pattern like $G_{2,1} = \{I_1, B_2, B_4, B_5, B_7, P_2, B_{10}, B_{12}\}$. The second slave node has a pattern of $G_{2,2} = \{B_1, B_3, P_1, B_6, B_8, B_9, B_{11}\}$. From $G_{2.1}$, there are the following data dependencies:

  - B_2, B_4 need I_1, P_1 as reference frames;
  - B_5, B_7 require frames of P_1 and P_2;
  - P_2 needs P_1;
  - B_{10}, B_{12} need P_2, I_2;

  With the union of all required reference frames (i.e., I_1, P_1, P_2 and I_2) and the elimination of those frames residing locally (i.e. I_1 and P_2), the remaining set (i.e., P_1 and I_2) consists of those extra reference frames needed to be transmitted to the first Slave node.

- By continuing the similar procedure, the extra reference frames for Slave node 2 are I_1 and P_2.

- Thus, the total communication overhead for $D = 1$ with $N = 2$ consists of 4 extra frames (e.g., about 2 Mbits from our test video file).

Similarly, we can calculate the extra reference frames for other N's. For example when $N = 3$, we have the frame patterns in three slave nodes as: $G_{3,1} = \{I_1, B_3, B_5, B_8, B_{10}\}$, $G_{3,2} = \{B_1, B_4, B_6, P_2, B_{11}\}$, $G_{3,3} = \{B_2, P_1, B_7, B_9, B_{12}\}$. The extra reference frames of each node are $\{P_1, P_2\}$ for node 1, $\{I_1, P_1, I_2\}$ for node 2, and $\{I_1, P_2, I_2\}$ for node 3. The total communication overhead now increases to 8 extra frames (e.g., about 4.5 Mbit) with a $D = 1$ and $N = 3$ configuration. Figure 4.1 depicts the calculated amount of extra communication when the values of

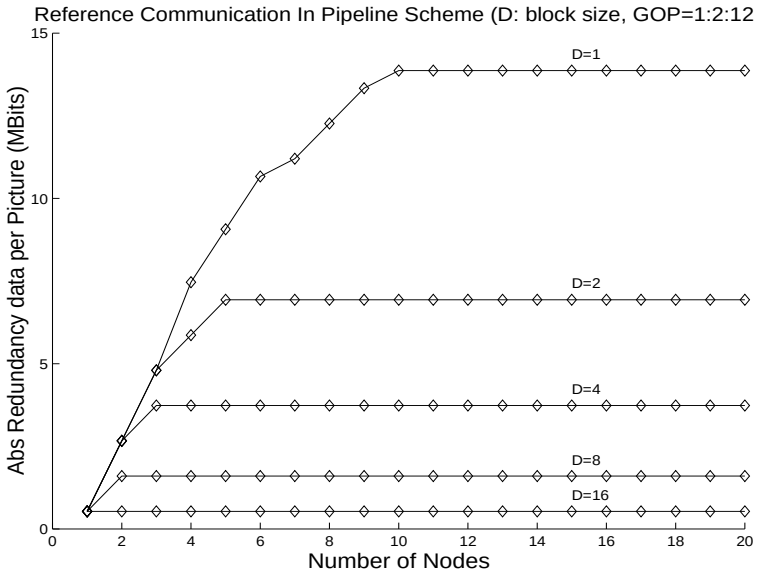

Figure 4.1: Communication overhead with different (D, N) combinations

N and D increase. We use the same GOP frame pattern of 1:2:12 as discussed in the above example. The frame size is 1024*1024 with one byte for each pixel, which results in one megabyte for each frame. The purpose of this analysis is to identify the performance trend of extra communication. Several trends can be observed as following:

- When $D = 1$ and N is relatively small, the extra communication increases rapidly with N increasing. It causes about 2.6-Mbits/per frame of extra communication when 2 nodes are used. Doubling the value of N from 2 to 4 will

increase the amount of extra communication from 2.6 Mbits to 7.3 Mbits (i.e., 280% increase). An additional 64% (i.e., from 7.3 Mbit to 12 Mbit) of the extra communication is expected when the number of nodes increases to 8. However, when the number of nodes increases from 8 to 16, it is expected that the extra communication will be saturated at 14 Mbit/per frame (i.e., 16.7% increase).

We believe the peak overhead on the extra communication is achieved since the GOP pattern will repeat itself among all these 16 nodes. When the number of nodes is greater than the number of frames within a GOP, only a small variation on the communication overhead is expected.

- When $D = 2$, a similar performance trend is expected. We expect a communication overhead of 2.6 Mbits, 5.7 Mbits, and 6.8 Mbits per frame when N is 2, 4 and 8. The degree of traffic increases is thus 120% and 20% . Less communication overhead is expected and the situation occurs earlier compared to the $D = 1$ case. This is feasible since larger D provides the opportunity to share the same reference frames within the same node, while it is infeasible when $D = 1$. For instance, a node that is assigned with two continuous B frames only needs one extra I- and P-frames.

- This trend is further depicted when $D = 4$, 8 and 16. Especially when $D = 16$, the minimum communication overhead could be approached. Furthermore, the communication cost remains unchanged when the number of slave nodes increases, because each node is now assigned a block consisting of a whole GOP and possibly the I-frame from the next GOP. Therefore, the extra communication for other reference frames is expected to be minimal.

- Given a fixed D, the derivative of the curve decreases when N becomes large. In fact, the curve always converges into a constant value when N is large enough.

- The increase of D can affect the extra communication significantly. While keeping the value of N fixed, increasing D can result in a reduction in extra communication cost.

  For $N = 8$, a 44% reduction is achieved if the value of D changes from 1 to 2. A further increase of D to 4 and 8 will produce an additional 46% and 56% performance gain respectively. Increasing the value of D means a bigger granularity, thus some B- and/or P-type frames can share the same reference frames within one slave node. Therefore, in terms of reducing the communication overhead, increasing the D value is quite effective.

The above examples are based on the GOP structure where I:P:B =1:2:12. For other GOP structure, a similar communication analysis can be conducted. For instance, in the case of I:P:B=1:4:10, we have more P-frames and less B-frames. The communication cost may decrease due to this change, since each B-frame needs two reference frames and each P-frame only needs one reference frame. Though interesting, it is beyond the scope of this work to further discuss how GOP structure affects the total communication overhead. Nevertheless, the general trends of communication with respect to block size D will hold even if a different GOP setting is

assumed. Specifically, when the block size equals to the length of GOP, a minimal communication overhead can be obtained in all cases.

4.2 Performance Analysis

To analyze the pipeline performance, we particularly focus on several events where Master and Slave nodes synchronize to each other. To simplify the discussion, we assume a homogeneous environment where each node is identical. As described in the previous sections, the system behavior could be discussed with the round concept which is observed in the Master point of view. Putting in a simple manner, a round is the period of time the master node experiences between two successive "polling" to a particular slave node. The round time should be determined by the decompression time of the interested slave node and the master node's interaction with other slave node, whichever is longer. With the homogeneous assumption, we can expect that each round (cycle) spends about the same time for each slave node after the pipeline run into a stable state. Thus, tracing the events happened in one round should be enough to demonstrate the overall system behavior. To describe the timing relation between different events, Table 4.2 lists some notations to be used. It is noticed that the following analysis of system timing is applicable to both cluster and SMP environments. This is, a 10-processor SMP machine with 1Gbps internal data communication bandwidth is treated the same as a 10-workstation cluster with Gigabit network. As an example, Figure 4.2 illustrates the events of the i^{th} round in the two slave nodes configuration. Our pipeline design put the majority of the the computation workload at the slave nodes. Nevertheless, certain part of computation has to been done in the master node, such as bitstream reading, parsing, and segmentation. Our experiments shows that this part only represents a tiny portion of the overall decoding time (less than 5% of total time). In the following performance analysis, we use a fixed value c to include this cost.

Table 4.1: Some notations used to describe system events for pipeline scheme

$T_{i,1}^{M}$	Master node starts to receive decompressed frames at round i
$T_{i,2}^{M}$	time when Master node finishes round i
$T_{i,1}^{S_k}$	Slave k starts to decompress at round i
$T_{i,2}^{S_k}$	time when Slave k finishes the decompressing of round i
T_{single}	average time needed for a single node to decode a frame
T_{sm}	time for data transmission from Slave to Master each round
T_{ms}	time for data transmission from Master to Slave each round
T_{round}	the round time for a complete decoding cycle
AS_f	average frame size in a MPEG-2 bit-stream
$R(D)$	the amount of reference frame data , w.r.t a given block size D
c	a small constant represents the misc software cost
T_p	the overall decoding time for a MPEG-2 stream
x	the number of frames in a MPEG-2 stream

We define $T_{i,1}^{M}$ as the beginning of the i^{th} round on the master side, when it starts to receive the results from the first slave. $T_{i,2}^{M}$ is defined as the end of the i^{th} round. Two conditions hold before the master can enter into the next round: (1) the master side has polled over all slave nodes for this round, and (2) the first slave node has completed decoding for this round. This is shown in Equation (4.1),

$$T_{i,1}^{M} = \max\{T_{i-1,2}^{S_1}, T_{i-1,2}^{M}\} \tag{4.1}$$

Here $i = 1, 2, ,$, is the round counter and N is the number of Slaves. The master's activity in the i^{th} round finishes after it sends a block of a raw frame to the N^{th} slave node, which is also the last slave. Thus $T_{i,2}^{M}$ and $T_{i,1}^{S_N}$ occur simultaneously.

$$T_{i,2}^{M} = T_{i,1}^{S_N} \tag{4.2}$$

The time-stamps at the slave side are depicted from Equation (4.3) to (4.6) as follows:

$$T_{i,1}^{S_1} = T_{i,1}^{M} + T_{sm} + T_{ms} + c \tag{4.3}$$

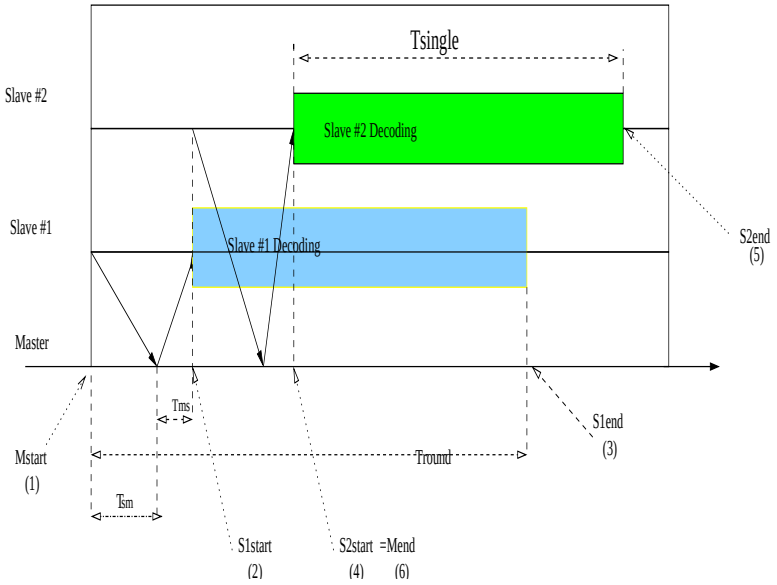

Figure 4.2: Events in one round with two slave nodes

At $T_{i,1}^{S_1}$, the first slave can start its decompressing work immediately after it receives a complete data set from the master. The delay is represented in (4.3) as the communication delay ($T_{sm} + T_{ms}$ for mutual data-exchange) and other software overhead ($2c$ assuming the same cost on both sides).

The decompression at slave node 1 needs $D.T_{single}$ seconds (Equation (4.4)) for D frames in average.

$$T_{i,2}^{S_1} = T_{i,1}^{S_1} + D.T_{single} \tag{4.4}$$

For the k^{th} slave node ($1 < k \leq N$), we define the start-time and end-time pair ($T_{i,1}^{S_k}$, $T_{i,2}^{S_k}$) in a similar manner. Since the master node polls the slave nodes in a round-robin order, node k will start later than node $(k-1)$. The delay time is $T_{sm} + T_{ms} + c$ (see equation (4.5)).

$$T_{i,1}^{S_k} = T_{i,1}^{S_{k-1}} + T_{sm} + T_{ms} + c \tag{4.5}$$

$$T_{i,2}^{S_k} = T_{i,1}^{S_k} + D.T_{single} \tag{4.6}$$

Equation (4.7) and (4.8) show the communication cost between master and slave nodes, which is determined by frame size $h.v$, network bandwidth B, and extra communication of reference frames $R(D)$. The average compressed frame size AS_f can vary according to different encoding parameters. We assume $AS_f = Frame\_Size/r$, r is the compression ratio (in our test stream, $r = 10$).

$$T_{sm} = \frac{D.h.v.8}{B} \tag{4.7}$$

$$T_{ms} = \frac{D.AS_f + R(D)}{B} \tag{4.8}$$

By substituting $T^{s_1}_{i-1,2}$ of (1) with (4), we have $(T^{s_1}_{i-1,1} + D.T_{single})$. Then $T^{s_1}_{i-1,1}$ can be replaced by $(T^M_{i-1,1} + T_{sm} + T_{ms} + c)$. Thus $T^{s_1}_{i-1,1}$ of (1) can be replaced by $(T^M_{i-1,1} + D.T_{single} + T_{ms} + T_{sm} + c)$. Similarly, $T^M_{i-1,2}$ of (1) is replaced by (2) at first, which is $T^{s_N}_{i-1,1}$. Then, by applying (5) k times, $T^M_{i-1,2}$ is finally represented in terms of $T^M_{i-1,1}$.

$$T^M_{i,1} = \max\{T^M_{i-1,1} + D.T_{single} + T_{ms} + T_{sm} + c, T^M_{i-1,1} + N.(T_{sm} + T_{ms} + 2c)\}$$

Therefore, the round time becomes

$$T_{round} = T^M_{i,1} - T^M_{i-1,1} = \max\{D.T_{single} + Tms + Tsm + 2c, N.(T_{sm} + T_{ms} + 2c)\}$$

It is now clear that the round time T_{round} is a joint effect of single-slave-decompression-time, total round communication time, and other system overhead. The total run time for a x-frame video stream can be expressed as:

$$\begin{aligned} T_p &= T_{start} + (\frac{x}{N.D} - 1).T_{round} \\ &= T_{start} + (\frac{x}{N.D} - 1).(\max\{N.(T_{sm} + T_{ms} + 2c), D.T_{single} + T_{sm} + T_{ms} + 2c\}) \end{aligned}$$

The start-up time T_{start} is the latency of the system accepting a user's request. It corresponds to the time period from the master's receiving of the first byte, to the first frame being displayed. The pipeline runs (number-of-rounds * T_{round}) seconds.

Here the number of rounds is calculated as $(\frac{x}{N.D} - 1)$, where x is the total number of frames . The average frame rate of decompression (FRD) can be estimated as

$$FRD = \frac{x}{T_p} = \frac{x}{T_{start} + (\frac{x}{N.D} - 1).T_{round}} \tag{4.9}$$

Equation (4.9) predicts the expected decoding frame rate. In order to have a clear view of system performance with respect to D and N, we further assume that the video sequence is long enough (i.e., x is large). Thus the start-up time T_{start} is negligible compared to the total decoding time. With (4.7) and (4.8), we are able to rewrite T_{round}.

$$\begin{aligned} T_{round} &= \max\{N.(\frac{D.AS_f + R(D) + D.h.v.8}{B} + 2c), \\ &\quad D.T_{single} + \frac{D.AS_f + R(D) + D.h.v.8}{B} + 2c\} \\ &= (\frac{D.AS_f + R(D) + D.h.v.8}{B} + 2c) + \\ &\quad \max\{D.T_{single},\ (N-1).(\frac{D.AS_f + R(D) + D.h.v.8}{B} + 2c)\} \\ &= \frac{D.(1+1/r).h.v.8 + R(D)}{B} + 2c + \\ &\quad \max\{D.T_{single},\ (N-1).\frac{D.(1+1/r).h.v.8 + R(D)}{B} + 2c\} \end{aligned} \tag{4.10}$$

Further observations reveal that the round time T_{round} is a nonlinear function of the block size D and the Slave number N. Both $(N-1).\frac{D.(1+1/r).h.v.8+R(D)}{B}+2c$ and $D.T_{single}$ increase when D and N increase. However, with a fixed D, the former still increases with N and the later does not change accordingly. When using less slave nodes (i.e., small N), the round time T_{round} will be dominated by the single-node decoding time T_{single}. This is explained by equation (4.11).

$$D.T_{single} > (N-1).\frac{D.(1+1/r).h.v.8 + R(D)}{B} + 2c \tag{4.11}$$

By replacing (4.11) into (4.10), the round time is further simplified as:

$$T_{round} = \frac{D.(1+1/r).h.v.8 + R(D)}{B} + 2c + D.T_{single} \tag{4.12}$$

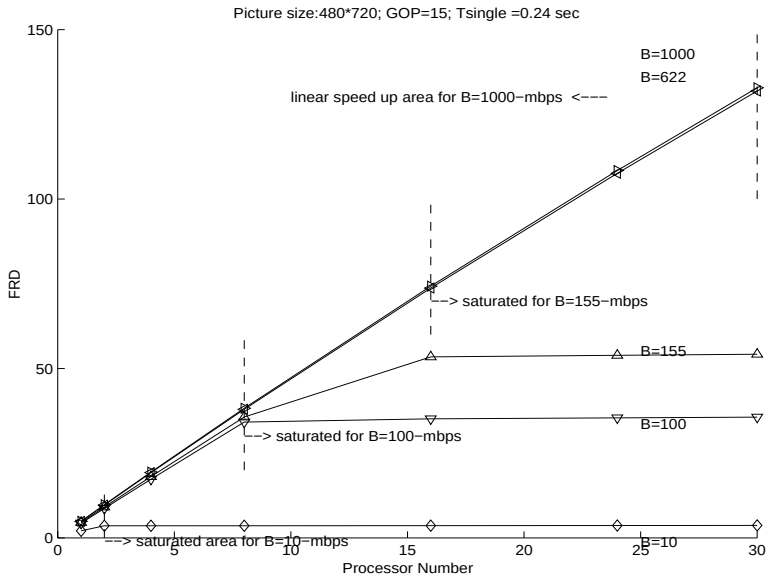

Figure 4.3: Expected performance of pipeline decompression scheme (with D equals one GOP)

With the simplified T_{round}, we can predict the system performance FRD. Figure 4.3 demonstrated our theoretical frame rate with respect to N and different network bandwidths. These network bandwidths include 10-Mbps and 100-Mbps switched Ethernet, 155-Mbps ATM OC-3, 622-Mbps ATM OC-12, and Gigabit Ethernet. We notice that the estimated FDR is linearly relates to N initially. For example,

- Using 10-Mbps switched Ethernet, our model predicts the FRD should increase from 3-fps to 6-fps when N is increased from 1 to 2.
- Similarly, with 100-Mbps switched Ethernet, the system expects the FRD can be reached to 36 fps with $N = 8$.
- We also expect, with 155-Mbps ATM OC-3, 55 fps can be reached by using 16 slave nodes.
- When the network bandwidth is more than 622-Mbps, the system expects an almost-linear improvement, which can possibly reach hundreds of frames per second.

However, the improvement is expected to be of much less significance after the above thresholds for these different networks. We realized that, when N keeps increasing, communication time will eventually dominate the round time. When

Equation (4.11) no longer holds, we have

$$D.T_{single} < (N-1).\frac{D.(1+1/r).h.v.8 + R(D)}{B} + 2c \tag{4.13}$$

and the T_{round} is changed to

$$\begin{aligned} T_{round} &= [\frac{D.(1+1/r).h.v.8 + R(D)}{B} + 2c] + (N-1).[\frac{D.(1+1/r).h.v.8 + R(D)}{B} + 2c]\} \\ &= N.(\frac{D.(1+1/r).h.v.8 + R(D)}{B} + 2c) \end{aligned}$$

Now the round time does not relate to the single slave decoding time (T_{single}) anymore. Accordingly, the frame rate becomes

$$\begin{aligned} FRD &\approx \frac{x}{(\frac{x}{N.D} - 1).N.(\frac{D.(1+1/r).h.v.8+R(D)}{B} + 2c)} \\ &\approx \frac{x}{\frac{x}{D}(\frac{D.(1+1/r).h.v.8+R(D)}{B} + 2c)} \\ &\approx \frac{B}{8.h.v.(1+\frac{1}{r}) + (\frac{R(D)+2c.B}{D})} \end{aligned}$$

Therefore, the frame rate will not increase even if more slave nodes are deployed (i.e., the number of slave nodes N did not appear in the above approximation). The system has reached its saturation point.

Due to the pipeline operation and the relatively heavy subtask in the slave nodes, the data pipeline scheme is expected to has a longer start-up latency T_{start} than the data-partition scheme. In the data pipeline scheme, a long waiting time is required to establish the system pipeline. According to the control algorithm in the master node, the following must be accomplished before the first frame can be displayed: (1) the master node sent out D blocks of compressed frame data to each of N slave nodes, (2) the first slave node finished the decompression of its assigned frames (i.e., from first frame to D^{th} frame), (3) the bitmap of decoded D frames were sent back to master node. Therefore we have:

$$T_{start} = \max\{N.(T_{sm} + T_{ms} + 2c), D.T_{single} + T_{sm} + T_{ms} + 2c\}$$

It is clear that the latency is also a function of (D, N) and other system parameters. Before the saturation point, it is determined by decompression time for D frames. Assuming $D = 15$, $T_{single} = 0.2$ seconds and network bandwidth is 100-Mbps, we expect a 3.5 seconds delay (for 720 * 480 video). When saturated, T_{start} will increase when N increases. This is caused by the enlarged duration of the pipeline cycle. The cycle time increases by $T_{sm} + T_{ms} + 2c$ seconds for each of the extra nodes beyond the saturation point. For the 100-Mbps network, this is about 0.48 seconds. For the 1000-Mbps network, we expect 0.05 seconds for each additional node.

With a more sophisticated design of the master and slave control algorithm, we can reduce the latency to $(T_{single} + T_{ms} + T_{sm})$. When the master node is distributing raw data to a slave node, it will receive decompressed image data from another slave node. When the pipeline is stabilized, slave nodes are running at different phase of decompression, thus maximize the parallel gain.

4.3 Experimental Results

In order to evaluate the data pipeline scheme and verify the correctness of the performance model, experimental results were collected from different hardware/software configurations. The testing environment for the data-partition scheme was used, we also added a 100-Mbps LAN equipped with low-end client nodes for more complete comparison. The experimental results indicate that our proposed system delivers a close-linear speed-up when high-speed networks are used. For example, with a 100-Mbps switched Ethernet, a 30-fps decompression is accomplished with 6 PCs (the corresponding speed-up is about 5). For higher display rates, we have observed up to 73-fps using a Sun SMP server with a network bandwidth of 680 Mbps.

4.4 Experiment Design

Table 4.2 listed the hardware/software configuration of the testing environments. The three test cases are labeled as NT100M, LINUX100M, and SMP680M respectively. Here, NIC indicates the speed of the network interface card, which might be different from the speed of the network switcher (as in the case NT100M). From our experience, the major factor determining the scalability of parallel decompression is the achieved network bandwidth. The achieved bandwidth is less than the peak bandwidth specified by the network or NIC hardware. Note that the achieved network bandwidth can be affected by the CPU speed, NIC and operating system. Thus, this observation indicates an end-to-end application-level measurement by combining all of the above factors.

Table 4.2: Experimental configurations and environments

System Parameters	NT100M	LINUX100M	SMP680M
Processor	Pentium 133 Mhz	PentiumII 450 Mhz	15 UltraSparc 248 Mhz
Memory	64 MByte	128 MByte	2 GByte
NIC	10/100 Mbps	10/100 Mbps	10/100 Mbps
Network Switch	100 Mbps	100 Mbps	100 Mbps
Operating System	NT4.0	RedHat Linux 6.2	Sun 5.6 SMP
Achieved End-to-End Bandwidth	80 Mbps	90 Mbps	680 Mbps
T_{single} (sec)	0.58	0.18	0.21

The 100-Mbps switched Ethernets (i.e., NT100M and LINUX100M) provide a basic environment for our experiments. The LINUX100M delivers a network efficiency of 90% . For the Sun SMP server, the actual communication between two processors is performed at the system bus level. The equivalent bandwidth is much higher than the network interface. This explains why the actual bandwidth (i.e., 680 Mbps) in the SMP system is much higher than its NIC speed. Note also that the CPU speed affects the T_{single} in a significant degree. A fast CPU results in a short

T_{single}. For instance, 450-Mhz Pentium II can reduce the decompression time (within a slave node) from 0.58 seconds (133-Mhz Pentium) to 0.18 seconds.

Our testing video stream is a MP@ML MPEG-2 bit-stream with 720*480 image resolution, encoded at an average bit rate of 6 Mbps. Our parallel decoder is a revised version, based on a public-domain sequential MPEG-2 encoder/decoder [26]. We only investigate the decoder part in this chapter. The inter-node communication is implemented by synchronous MPI (message passing interface) protocol. Although asynchronous message passing could be used to improve the network efficiency, the potential benefit may be marginal since our experiments are performed in the clear environment.

We measure the following performance metrics from each experiment:

- FRD: The actual frame rate of decompression, which is compared to an expected frame rate from our analytical model.
- Speed-up: This is a scalability measurement on the achieved FRD when the number of slave nodes is increased.
- CPU usage: The utilization of the CPU in each slave node.

4.5 Performance over A 100-Mbps Network

Figure 4.4 shows the performance results for the NT100M environment. Note that the slave nodes are equipped with 133-Mhz Pentium. If used alone, it is only capable of decoding at 2-fps when using the sequential decoder. With our proposed software solution we have demonstrated that the decompression rate can be scaled to at least 15-fps. The 15-fps performance usually produces an animation-like quality that video is continuous (instead of 2-fps slide-show quality). Detailed analysis on the performance results are the following:

- Our experimental results in Figure 4.4 show a close-linear improvement of decompression rate when the slave nodes increase. For one slave node, we observed 1.8-fps. When increased to two slave nodes, a 3.5-fps decompression rate is measured (with a speed-up of 1.94). Further increase to 4 slave nodes results a 6-fps decoding rate. (about 3.33 times higher). At 8 nodes, the system can deliver a throughput of 14-fps, this represents a 7.78 speed-up.

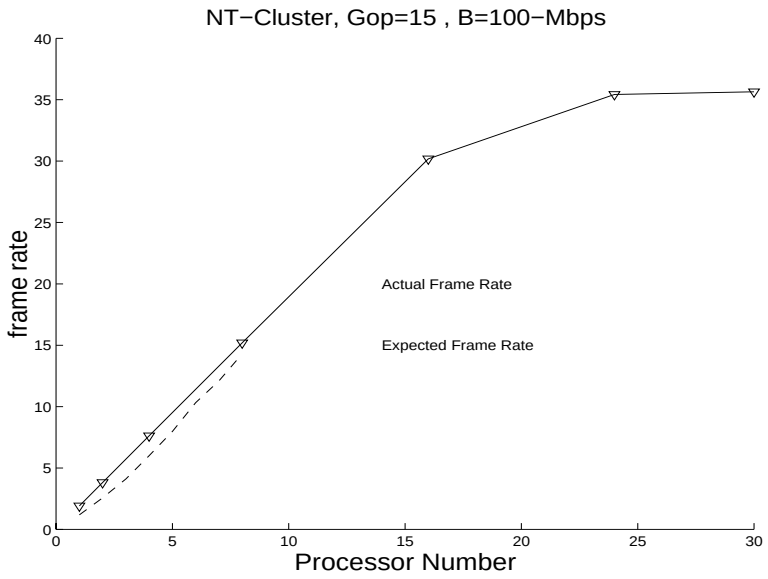

Figure 4.4: Pipeline decoding experiment on a cluster of Pentium 133 PC workstations with 100-Mbps fast switched Ethernet.

- The actual system performance conforms with the analytical model closely. For 1, 2, 4 and 8 slave nodes, there are only 11%, 12.5%, 14% and 6% differences between the expected and observed decompression rate. The small differences between the analytical and experimental results indicate a great consistence on the system behavior before the saturation point (as we pointed out in the last section).

The expected frame rate from our model is 35-fps, which can be approached with 24 slave nodes according to our prediction. Due to the lack of sufficient slave nodes in the NT100M platform, we were not able to verify the behavior after this performance saturation point (e.g., N=16 or 24). Fortunately, another set of experiments performed on cluster of Linux machines (i.e., LINUX100M environment) to give us more insight.

Within the LINUX100M configuration, each node is a PentiumII 450-Mhz PC. The single-node decoding speed is 5-fps. The highest frame rate predicted under this cluster environment is 36-fps, at an 8-slave-node configuration. Detailed results are discussed as following:

- Again, we observed that the decoding rate increases close-linear when the number of nodes grows from one to five. For 1 slave node, we have 4.2-fps. Two

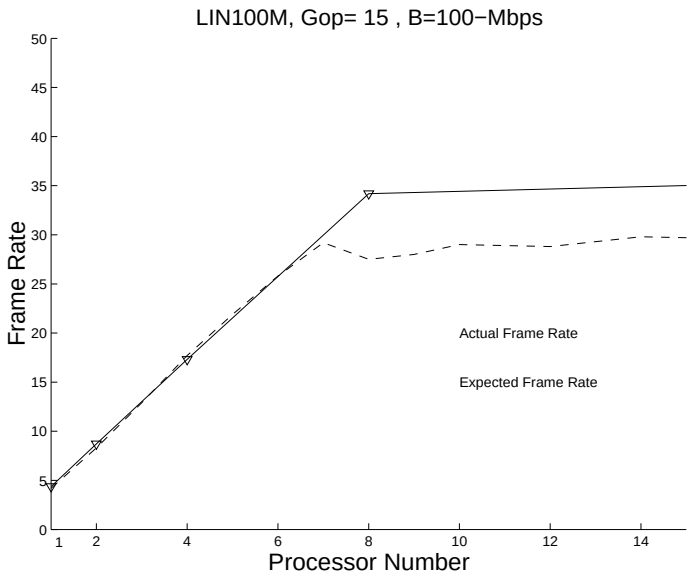

Figure 4.5: Pipeline decoding experiment on a PentiumII PC Linux cluster with 100Mb fast switched Ethernet

slave nodes produces 8.3-fps, which is a 97% improvement. Further increasing to 4 slave nodes brings frame rate to 21.5-fps, nearly 4 time higher than one-slave node case.

- (saturation point) The system reaches its peak performance at 7 slave nodes, with a frame rate of 30-fps. Then the frame rate becomes flat, with only a tiny fluctuation along 30-fps. In fact, the performance gain becomes insignificant after 5 slave nodes. The saturation point comes earlier than expected (e.g., the analytical model predicts the system saturation point at 8 slave nodes.)
- Nevertheless, the observed results still show an acceptable match with the analytical model. For one and two slave nodes, the mismatches are less than 5%. After saturation point, the observed throughput is about 14% less than expected (e.g., the analytical model predicts 35-fps, while the experimental result is 30-fps after saturation).

The LINUX100M configuration along with our proposed parallel software decoder represents a reasonable solution for providing a real-time MPEG-2 decompression. The achieved quality can be as great as any hardware-based solutions since it achieved up to 30-fps. However, if the system is allowed to support flexible user interactions (e.g., fast motion display) or 60-fps HDTV video quality, decompression with a higher-than-30-fps rate is required.

Due to the network bandwidth limitation, LINUX100M is not capable of providing this flexibility. Therefore, we investigated the performance over the 680-Mbps

Sun SMP platform. Since we used MPI as part of the communication mechanism, our proposed software solution did not require any modification on the SMP environment. Note that this kind of high bandwidth is not easily achieved in today's networking environment. Early results in Gigabit Ethernet just started revealing similar performance results in the range of 600 Mbps over high-end servers [27].

4.6 Performance over A 680-Mbps SMP Environment

Figure 4.6 is our high-performance result using an SMP machine with 15 UltraSparc 248-Mhz CPUs. As mentioned in Table 4.2, the inter-process communication is 680-Mbps. We tested three video titles with different content: Tennis, Flower and Mobl. These three videos are encoded with same encoding parameters to have a fair comparison. They are chosen to represent different degree of image complexity and object motion. Nevertheless, our experiments are quite consistent for all the testing stream. The difference in decoding performance for these video titles is almost negligible. Part of the reasons of this performance consistency comes from the high-encoded-bit-rate. With 6Mbps encoding bit rate, few macroblocks are skipped, thus the number of macroblocks decoded for the 3 video streams are very close to each other. This is not the case when the encoding bit rate is small, where many macroblock is skipped during encoding. In the following we only discuss the performance for the tennis. We have the following observations:

- 30-fps real-time decompression rate is achieved at 7 slave nodes. With 13 slave nodes, the system is able to provide 60-fps HDTV quality. The highest measured system performance is 68-fps, using 14 slave nodes out of the total of 15 physical CPUs.
- The actual frame rate also indicates a close linear speed-up. Starting from 5-fps at one slave node, we observe 10-fps at 2 slave nodes, implying a 100% speed-up. The speed-up for 4,8 and 14 slave nodes are 400%, 780% and 1360% respectively. This indicates that our pipeline scheme can be well scaled up for a high-demanding video decompressing scenario. After 14 slave nodes, the decompression rate becomes flattened (not shown in the plot). Further increasing the number of slave nodes could not bring more performance gain, simply because all of the 14 CPUs have been fully loaded.
- The system shows a precise behavior as predicted by our analytical model. The deviation of decompression rate is controlled within a 10% error range.

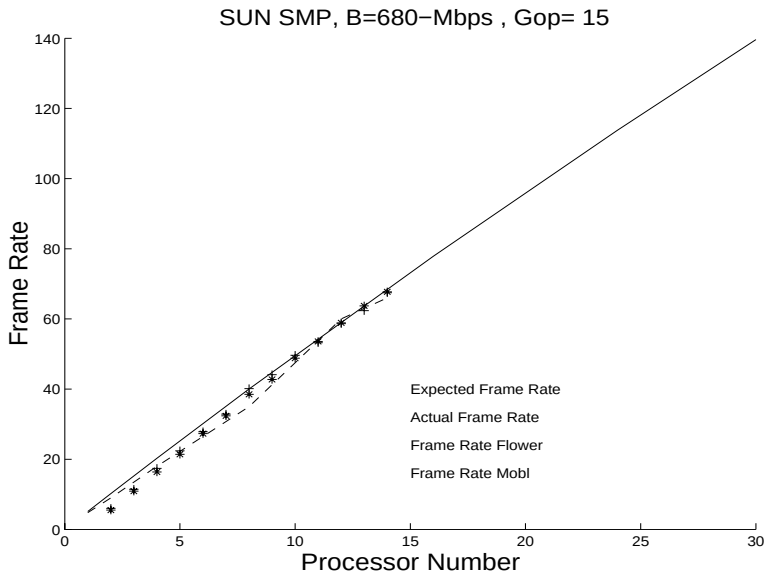

Figure 4.6: Pipeline decoding experiment on SUN eclipse server with 680Mbps sustained bandwidth.

> According to our analytical model, the system should saturate at 55 slave nodes, which should possibly provide a 270-fps decompression rate.

Table 4.3 shows CPU usage for the master and slave nodes. In the SMP machine, the slave processes are dynamically scheduled to 14 physical processors by the operating system. Thus the statistics is collected directly from the master and slave nodes. It is observed that the slave nodes keep a high CPU utilization through all the experiments. In average, 90% of CPU time in slave nodes is used in the user space for computation, and the rest of computation is spent on communication and miscellaneous system cost. The system also runs in a highly balanced manner, and the standard deviation of slave node load is very low (less than 7%). The waiting time in slave nodes is also controlled in a low level, ranged between 5% and 8%.

Unlike the case of data-partition scheme, where the overhead in master node become significant at high system configuration, the data pipeline scheme has low master node complexity. The computation in the master node is kept low (15% at two slave node), and increases slightly when more slave nodes are adopted (refer to the Master Load row in Table 4.3). This indicates that the computation overhead

in master node is not significant. At the 14-slave-node configuration, there is still 70% idle time in the master node, indicating the master node is not yet saturated. Further performance improvement could be obtained when more than 14 slave nodes is used.

Table 4.3: CPU utilization of master and slave nodes

	Number of Slave Nodes							
Parameters	2	3	4	6	8	10	12	14
Slave CPU Load (Average)	92%	89%	90%	91%	90%	90%	90%	91%
STD of Slave Load (Average)	7%	6%	6%	5%	6%	4%	3%	4%
Slave Waiting (Average)	5%	5%	6%	7%	7%	6%	8%	8%
Master Load	15%	18%	20%	22%	25%	28%	30%	32%

The improvement of the decoding performance is further verified by the cumulative system CPU utilization. With two slave nodes, 10% of the total computing power of the 15-nodes SMP machine is used. The number becomes 16% with 3 slave nodes, and increases by about 7% for each additional slave nodes afterward. With the full configuration of 14 slave nodes, 90% of overall processing power is used by the parallel decoder, and remains at this level when further increasing the number of slave processes.

4.7 Towards the High Resolution MPEG-2 Video

The achieved frame rates for the low and main level MPEG-2 video are very close to our prediction. However, the scalability performance results for the high resolution MPEG-2 videos are not satisfactory. In Figure 4.7, the decoding rates for (1404*960) MPEG-2 files are illustrated. Starting with 2 fps at single node configuration, a linear increase can be observed. However, the decoding performance of "flower" suddenly dropped to 2.5 fps at 10 slave nodes, and continued deteriorating

with a small rebound at 11 slave nodes. For "tennis" and "calendar", similar performance degradation is observed at 12 slave nodes, right after the peak performance point. Similar performance degradation is observed for the (1024*1024) video format.

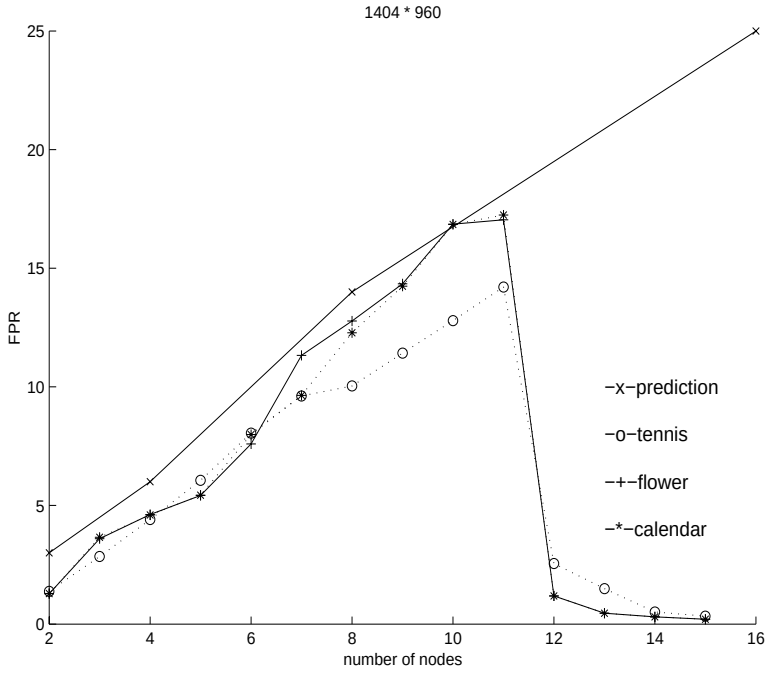

Figure 4.7: Decoding frame rate for 1404 x 960

In order to identify the system bottleneck which causes the degradation of decoding performance for high-resolution video, we record the utilization of system resources during the decoding process. The evidence from system runtime statistics can be collected from the CPU time distribution and the number of page faults to support this unique observation[1]. Figure 4.9 illustrates the measured number of page faults versus the number of slave nodes and the CPU statistic. For the sake of brevity, we only present the results for "tennis".

For the 352x240 video, the number of page faults virtually remains unchanged, and is kept at a low level (1010 page faults/frame). Increasing the video resolution to

[1]The CPU utilization could be obtained by system call *time()* in UNIX system, and the page fault is recorded by a utility process *truss* spawned by the decoding processes

704x480 is reflected by a rise of the page fault number, a four fold jump is observed. Nevertheless, the 704x480 case still has a flat curve for the increasing slave node, indicating that the system is running steadily. For the 1024x1024 video, the number of page faults increases considerably. It is noticed that the page faults significantly increases at 10 to 12 slave nodes, reaching 3500 page faults per frame. Compared to the decoding performance in Figure 4.7.(b), the period with high page faults coincides with the collapse of the decoding rate. This indicates that the excessive page faults had driven the system into an thrashing state. The page faults behavior of the 1404x960 video shows the same pattern as in the 1024x1024 case.

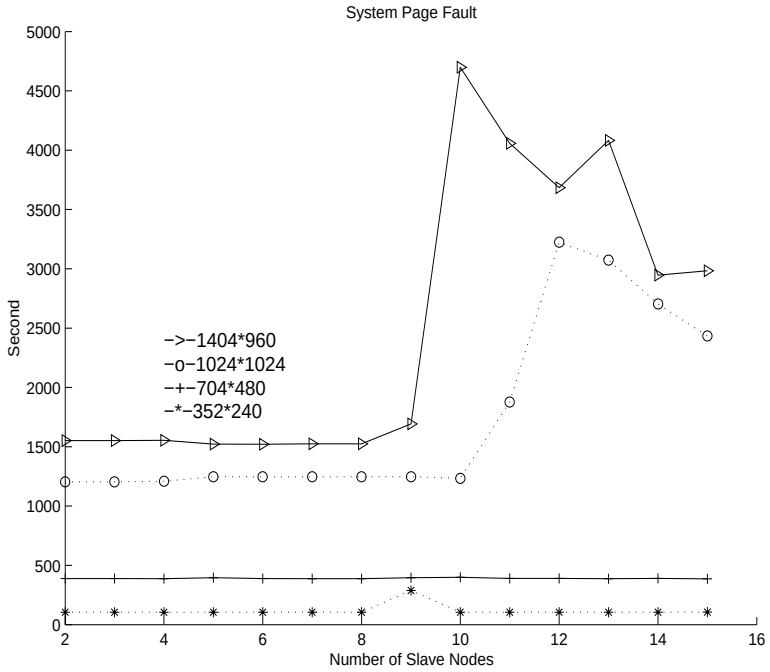

Figure 4.8: Page fault vs number of slave node

The excessive increasing of page faults is also indicated by the CPU usage. With one slave node, 90% of the system time is idle, 8% of the CPU time is used in the user space, and the remaining 2% for other system maintenance. When increasing slave nodes, the user space time increases proportionally, and the system idle time decreases. After 8 slave nodes, however, both system idle time and user space time drop significantly, while the system overhead shows a major increase. About 90% of

the CPU time is used by the operating system, while user space only occupies 5% of CPU time. Recalling that the page faults number increases suddenly at 9 slave nodes (see Figure 4.9), we conclude that the system spends most of its CPU time swapping page in/out, thus drops the decoding performance.

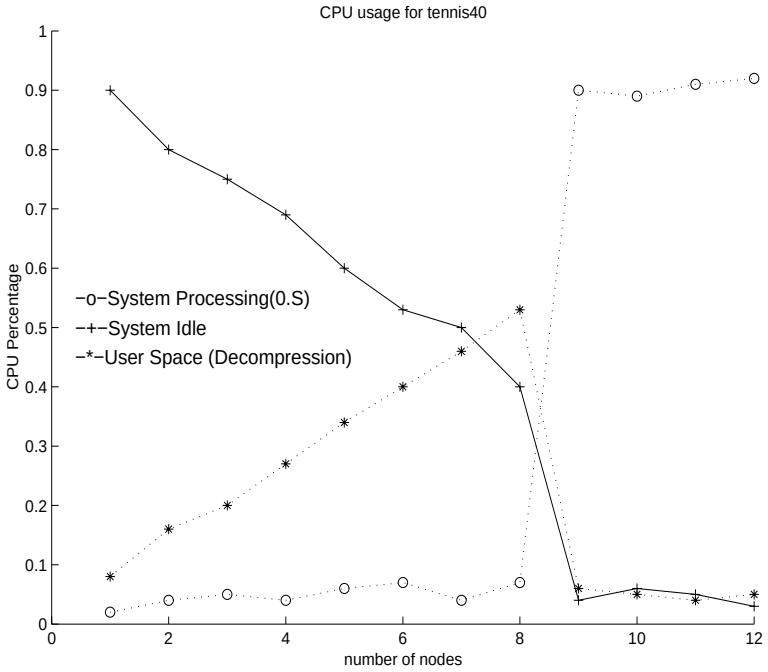

Figure 4.9: CPU usage

Realizing that page fault is directly related to the shortage of system memory, we believe that the buffer management of the parallel decode should be investigated. A not-optimized buffer scheme will devastate the competition between user processes (e.g., our communication and decompression software) and system processes (e.g., demand-paging mechanisms by OS). Because of the shortage of the overall memory, the system process will generate a significant number of page faults, which in turn slows down the decompression speed due to the lack of CPU.

The memory requirement for the master node and slave nodes can be expressed:

$$\begin{aligned} M_m &= m_c + m_{streambuffer} + m_{outbuffer} + m_{inbuffer} \\ M_s &= m_c + m_{compressedbuffer} + m_{transmissionbuffer} \\ &= m_c + m_{outbuffer} + 1.5 * m_{inbuffer} \end{aligned}$$

Here m_c is the size of executable code for the master node, about 500 KB. $m_{streambuffer}$ is the streaming buffer to receive the compressed video packet from the video server, we currently fixed it to be 1 MB. $m_{outbuffer}$ and $m_{inbuffer}$ are dedicated for information exchanging in the parallel decoding. $m_{outbuffer}$ equals one GOP of MPEG-2 compressed frames, and $m_{inbuffer}$ needs to accommodate two GOP of decompressed frames (one GOP for displaying and another for incoming traffic).

For the test stream tennis40 (1404*960), the corresponding memory requirement in master/slave sides are: $M_m = 42$ (MB) and $M_c = 30.8(MB)$. The accumulative buffering space will grow quickly when using a large-scale slave node configuration, which causes unsatisfactory scalability performance when the number of slave nodes is large. For instance, let N be the number of slave nodes, the total memory requirement becomes

$$M_t = M_m + N * M_s$$

Using the parameters of the testing MPEG-2 video, the total memory used can be estimated from above equation. For the video *Tennis40*, we need about 73 MB, 104 MB, 165MB, 319 MB and 381.5 MB respectively when the number of slave node are 1,2,4,9 and 11. In the next section, we will discuss several techniques to reduce the buffer space.

4.7.1 Efficient Buffering Schemes

It is noticed that the slave node allocated a GOP length of frame buffer originally, which could be further optimized to the minimal buffer space. However, due

to the decoding dependency inside the MPEG-2 video structure, we are not able to use only one frame buffer. To decode a B-frame, we need two reference frames and one decoding working frame, resulting in a total of 3 frames. With a careful redesign of the master-slave communication protocol, using a 3-frame transmission buffer in the slave side is possible, which we called the **ST** scheme. When the picture size is 1024*1024 and GOP=15, we save about 12 MB buffer space per slave node, about an 80% reduction in the slave side.

The minimum required frame buffer can be further decreased from 3-frames to 2 frames. For the I- or P- frames, we need one buffer for the prediction picture, and another buffer for the working frame. The two buffers change their role after decoding a I- or P- frame, so that the most recently decoded I- or P- frame is used as the prediction frame for the next P- frame. For the B- type frame, since the decoded B-frame will not be used as reference frame, we can directly send decoded blocks to the master node without storing them. The above discussion assumes that the reference frame for a current P- frame is always the last decoded P- frame, and the reference frame for a current B- frame are the last two P-frames. Nevertheless, this approach also works if the B- and P- frames always refer to the I- frame with corresponding change in the reference buffer.

With this scheme, the expected memory requirement becomes

$$\begin{aligned} M'_t &= M'_m + N * M'_c \\ &= M_m + N * (m_c + 1.5 * 3 * m_{frame} + m_{inbuffer}) \end{aligned}$$

To find out the number of maximum slave nodes before the exhausting of system memory for the 1404*960 single layered MPEG-2 video, we solve (47 + (N-1) * 6)<300 MB (use 300 MB as a system threshold). This gives N=43 slave nodes and a higher than 60 fps decoding rate can be expected.

4.7.2 Further Optimization in the Slave Nodes

It is further observed that the decoding procedure in slave nodes might not use three frames all the time. More specifically, the I-frame needs only one frame buffer, while P-frame can be decoded with two frame buffer. Only B-frame needs the whole three frame buffers. Thus the total amount of buffer can vary during the life time of the slave node. By allocating frame buffer dynamically according to the frame type, it can be expected that the total buffer can be significantly reduced for high quality video.

This is particularly true when I- and P-frame represent a considerable portion of the frames. Let the ratio of I, P, B frames in a GOP structure be a:b:c, the effective buffer space for one layer is expressed by $M = (1 * a + 2 * b + 3 * c)/(a + b + c)$. In a typical GOP structure of "IBBPBBPBBPBBPBB", we have a:b:c=1:4:10. This results in an effective buffer number of 39/15=2.6, which is about 85% of the 3 frames buffer scheme.

The concept of dynamic buffer allocation can be applied inside the decoding of each frame. Since the decompression of each frame is based on a serial decompression of macroblocks, the overall buffer space could be reduced by dynamically allocating buffer for macroblocks. For example, when decoding the first macro-block, we only need to allocate a 16*16 block space. The buffer for other macroblocks will be assigned when it is needed. With this dynamic memory allocation, we expect an additional buffer reduction of 0.5 frame for the working frame. Notice that this scheme can not reduce the amount of buffer for the reference frame, which should be in system during the decoding process. The effective buffer requirement becomes $M = ((1 - 0.5) * a + (2 - 0.5) * b + (3 - 0.5) * c)/(a + b + c)$. Using the same GOP structure as the above, the effective frame number of the buffer in slave node is 2.1, which is 60% of the 3-frame buffer scheme.

The dynamic allocation of buffer in the slave node is an application level memory management scheme, which is closely embedded in the decoding process. The current implementation rely on some system-provided routines (e.g., *malloc* and *free*). We speculate that a customized buffer management routine (direct access of the system memory) should be able to further increase the decoding performance, which should be discussed in our future research. The tradeoff here is the additional CPU cost introduced by the dynamic memory management. For each macro-block, the additional cost includes at least two system calls (for memory allocation/deallocation) and some other miscellous operations. It has been shown that the cost associated with dynamic memory allocation is significant for the database server and Web-server, where thousands of processes may co-exist to process user requests. In our case, the number of slave nodes/processes is usually below 20 and it is expected that memory management activity is far less frequent, thus the overhead introduced should be limited. This is confirmed by our experimental results by comparing the performance of the decoding with/without dynamic memory allocation. With dynamic buffer allocation enabled, the overall decoding time is increased less than 7% than the static memory allocation case.

4.7.3 Implementation and Experimental Results

Experiments are performed for the high resolution video format with the revised memory management scheme. Our results show that the two improvements bring significant memory reduction, and the phenomena of paging panic is eliminated.

For the 1404*960 video, the total buffer size is 53.5 MB with one slave node, which is 27 % less than the original one. For 4-slave-node case, the ST scheme use 85 MB instead of the original 165 MB, which is almost 50% memory saving. For the 1404*960 case, the number of page faults shrinks from 1500 to 1200, at one slave node configuration. For 1024x1024 video, the page faults are now 943, 25% less

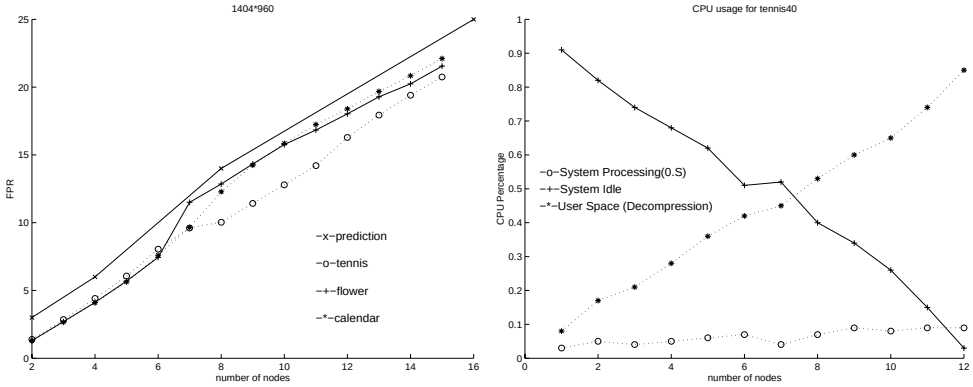

Figure 4.10: (a)Decoding frame rate for the revised memory management, and (b)user space time VS kernel space time for the first buffer optimization scheme

than before. For all of the video streams, the number of page faults almost remains unchanged when increasing the number of the slave nodes.

Figure 4.10.(a) show the scalable decoding frame rate for the 1404*960 video with our revised ST scheme. We observed a close-to-linear increase of the frame rate. The peak decoding rate is obtained at 14-slave nodes, where 20 fps is observed. The decoding performance for 1024*1024 video files shows a similar behavior. Thus our revised buffering scheme has successfully solved the memory shortage problem, and works well for high quality video up to MP@HL video.

Figure 4.10.(b) shows the overall CPU time distribution of slave nodes when decoding high-resolution video formats with the revised buffer scheme. The user space time component represents the computation time for the MPEG-2 decoding procedure, the kernel space time is for the system level overhead, including time spent in the network layer, system call, and other costs. It is observed that the user space time increases linearly when the number of slave nodes increases, accompanied by a corresponding decrease in the system idle time. Meanwhile the operating system level cost is maintained at a low level (between 5% to 10 % of total CPU time). For the large scale experiments (more than 11 slave nodes deployed), the abnormality cross-over of the user space time and system overhead observed in the original decoding

experiments no longer exists. This further proves the effectiveness of the **ST** scheme in solving the memory shortage.

4.8 Summary

Up to this chapter, we discuss how a generic and scalable MPEG-2 decoder is implemented via a pure-software-based parallel decoding scheme. The MPEG-2 decompression algorithm is parallelized in data-partition and data-pipeline manner built on a master/slave architecture. The *data-partition scheme* is shown not as scalable as the *data pipeline scheme*, due to the overhead in master node and high-bandwidth requirement. The *data pipeline scheme* is able to produce high gain from parallel processing with little overhead. We analyzed the effect of different block sizes and the speedups with increasing number of slave nodes. Using the block size of one GOP, the communication overhead is reduced to minimum by reducing the inter-frame dependence as much as possible.

Promising results show that *data pipeline scheme* performs well in various hardware/software platform, given necessary network bandwidth. The reported highest decoding rate is more than 70 f/s for ML@MP video. We further investigated the scalability performance for a wide range of video format, especially for the high-resolution MPEG-2 video (e.g., HDTV). However, It is found that the original data-pipeline scheme suffers significant performance degradation when decoding high-level MPEG-2 video with the full system configuration, due to inefficient management of memory space in the decoder. The shortage of system memory is also confirmed by the outbreak of page fault when system is fully loaded. We propose an efficient buffer management mechanism such that the memory requirement can be reduced by 50%. The revised parallel decode significantly relieve the memory shortage problem, and showed a satisfactory scale-up performance when decoding the high-resolution video formats (close to 24 f/s decoding rate is achievable for high resolution quality video).

Our analysis imply that upto 270 f/s is possible by increasing the number of slave node in the SMP environment.

Our investigation on parallelizing MPEG-2 decoding algorithm indicates that real-time decompression of high quality video can be obtained via pure software solution. We also believe that computation power in the future will be improved significantly via cost-effective MPP technology. Therefore, video decoding capability at the end-user could be regarded as solved.

In the following chapters, we discuss another critical link in distributed multimedia system— how to provide quality-guaranteed communication service in wireless network. As mentioned in chapter 1, delivering multimedia content timely with high fidelity in wireless network presents a great technical challenge, not to mention that the system should also support large number of users. Among several competing technologies (e.g., FDMA, TDMA, and CDMA), we choose WCDMA wireless network as the target platform, due to the many advantages CDMA have over other systems. In the next chapter, we discuss a dynamic spreading factor scheme in WCDMA system such that multimedia traffic with strict BER requirement can be supported by dynamically updating its spreading factor. The protocol provides guaranteed QoS for all accepted call requests. Our discussion also includes a time analysis for the protocol execution, and a traffic scheduler utilizing the dynamic spreading capability.

In chapter 7, we further extend the dynamic spreading factor scheme into the multiple cells environment. Specifically, we addressed how soft handoff algorithm in WCDMA should be revised to work with dynamic spreading factor control. Our studies show that the decision of spreading factor in handoff period should be considered together with the power control for mobile stations. With a heuristic algorithm to optimize the spreading factor and power control, we will show that the overall throughput during the handoff period can be improved by 25%.

CHAPTER 5
MULTIMEDIA SUPPORT IN CDMA WIRELESS NETWORK

Providing multimedia service through wireless network is becoming the major battle field for the service provider and technology venders. Recent technology advances are increasing multimedia capabilities in mobile devices. Cellular phones and notebooks are converging into a single mini-device which is capable of both computing and communicating. They are becoming more competitive to desktop PC in terms of computing power. However, the communication quality supported by current wireless network still need major improvement to meet the rigorous demands of multimedia applications, where higher data rate and lower bit error rate is desired.

In traditional wireless cellular network, traffic channels are designed to support voice conversation. These channels have the same data rate, and have the same bit error rate (BER). These systems only have limited support for data traffic, such as in Pour and Liu [28] where the silent period of voice is utilized for data transmit. Thus, providing quality of guarantee service in wireless network needs new design and functionality in the MAC layer [8]. Choi and Shin [29] discussed QoS guarantees in a wireless LAN with Dynamic Time-Division Duplexed (D-TDD) transmission. In paper [30], an admission control protocol for multi-service CDMA is developed based on interference estimation. They used log-normal distribution to approximate the effect of random user location, shadowing, and imperfect power control.

A major effort toward multimedia supports in the cellular wireless network is so called the 3rd generation system, such as WCDMA. CDMA based system is particular appealing for multimedia application, primary due to its capability to providing different level of link quality (BER) for different traffic type, thus make it

possible to better utilize the radio resource. The spreading factor of CDMA is the key variable in determining user data rate and associated BER. Theoretically, spreading factor makes CDMA possible by repeating user's data signals such that they can be re-constructed at the receiving mobile stations. Increasing spreading factor can benefit BER because it will increase the desired signal strength linearly. The mean MAI (Multi-Access-Interference) caused by other users will decrease accordingly, and will approach to zero when the spreading factor approaches to infinity.

The evaluation of the BER has been studied intensively [9, 31–33]. It is widely agreed that the BER is largely determined by MAI. In Fukumasa et al. [9], the design of PN sequence is discussed to reduce MAI. In Geraniotis and Wu [33], the probability of successful packet transmission is analyzed for DS-CDMA system. In Choi and Cho [34], a power control scheme is proposed to minimize the interference of high data-rate users to the neighboring cells. The result shows that the number of high data-rate users for data communication should be less than 6 in order to support enough voice users. However, these works did not cover the system BER with dynamic spreading factor. A new middle-ware protocol is needed to integrate different services more efficiently (i.e. voice, data, video).

The unique BER behavior and its correlation to spreading factor in CDMA system allow additional flexibility in transmitting data traffic. Akyildiz proposed a packet scheduling protocol for slotted CDMA [35]. The scheduler can maximize system throughput based on BER requirement. In Oh and Wasserman [36], system performance of a DS-CDMA when setting to a different spreading gain is studied. Two kinds of traffic are considered. The author shows that optimal spreading gain increases linearly when the MAI increases. However, their work did not relate the control of dynamic spreading gain with the system load.

To have a clear picture of how spreading factors and the number of active users affect the BER, we performed link level simulation for the asynchronized uplink

channels in IS95. Figure 5.1 depicts the BER under different numbers of active users and spreading factors. The experiments were performed within a single cell without interference from neighboring cells. Multi-path effects and thermal noise are not taken into account since we emphasize on the effect of spreading factor.

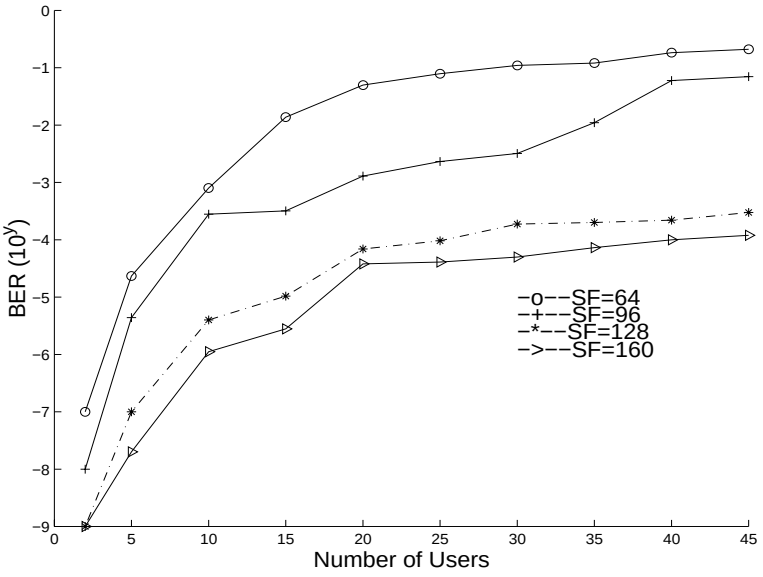

Figure 5.1: BER vs. user number vs. SF

The performance results clearly indicate that the increase of spreading factors can effectively decrease the BER for a given number of users. For example, with 10 users, increasing the spreading factors from 64 to 96 will reduce BER from 0.0008 to 0.0003 (e.g., 62.5% reduction). Increasing spreading factors further to 128 results a BER of 0.000004, which is 75 times less. In order to support a variety of BERs, many hardware manufacturers are considering to bring in the dynamic capability of changing spreading factors in next-generation mobile and base stations.

Because the possible adaptation of spreading factors, novel admission protocols (i.e., state diagrams) are proposed in mobile and base stations. With the new protocol, it is possible that a mobile user is notified to change its spreading factor.

This usually happens when a new mobile presents an OPEN request. The preliminary results indicate that our proposed system always maintains a desired BER for all the connections (including the existing and newly-arriving one).

Starting from this chapter, we present our on-going study for the new protocol design in WCDMA to support multimedia traffic and the associated performance issues. Our discussion in the first part of this chapter will focus on the baseline protocol design in a single cell situation. We also provide a detail analysis for the timing components when the protocol is executed and propose improved schemes to reduce the end-to-end connection setup time. In the second part of this chapter, we address the multimedia traffic scheduling schemes based on the new dynamic spreading factor protocol. In addition to guaranteeing the voice communication, our proposed scheduling scheme reduced the turn-around time significantly for the conventional data traffic (i.e., e-mails). These discussion can also be found at [37, 38].

The dynamic spreading factor protocol is further extended to handle multiple cells in the next chapter. We add handoff capability into the dynamic spreading protocol such that mobile station can take advantage of the dynamic spreading even during handoff period. To optimize the overall system throughput, we propose a new resource allocation algorithm to assign spreading factor and transmitting power to handoff mobiles. The majority of the content is also reported in references [39] and [40].

5.1 Performance-Guaranteed Call Processing

The ultimate goal of admission control protocol is to support as many users as possible while still satisfying BER requirements for all existing connections. Conventional FDMA schemes divide the frequency spectrum into multiple channels. Every user's connection needs to allocate one channel from the base station before the voice

conversation can take place. When all the channels are allocated, no more connections can be admitted. Therefore, admission control with FDMA schemes is straight forward.

However, CDMA-based schemes (along with the multimedia support and dynamic spreading factors) make the admission decision nontrivial. One advantage of CDMA system is that the whole spectrum is used for communication. Connections are separated by Pseudo-Noise codes (PN) assigned at the base station. Thus determining an exact point to block newly-arriving connections are difficult. During a typical call life time of voice conversation, the system will accept and terminate a number of calls, thus the number of active users will change rather frequently. In an interference-limited CDMA system, the variation of system load is the key factor determining the fluctuation of the link quality of traffic channel. Thus our admission control protocol needs to be adaptive to the changes in the system load. Our proposed admission control protocol is able to monitor and assure that performance remains almost the same by taking necessary steps whenever needed.

Both mobile and base stations need to participate in the call admission process. Mobile stations are the ones that make the requests and/or update their parameters following various commands from the base station. Base station should process requests from mobile stations, monitoring the change in the environment, and decide the key transmitting commands to mobile stations(such as power command, spreading factor, and PN codes). The environment change includes increase/decrease in the number of users and alterations in traffic types. As an example, a station may start a connection with voice, halt the voice without terminating the connection, send email, and go back to voice communication. This scheme eliminates the significant overhead (in term of tens of seconds) caused by terminating and re-opening a new connection. Therefore, the admission control protocol that we propose here is flexible and capable of guarantee overall system performance.

5.1.1 Proposed Admission Protocol in A Mobile Station

The following Figure 5.2 depicts the state diagram (i.e., protocol) for the mobile stations.

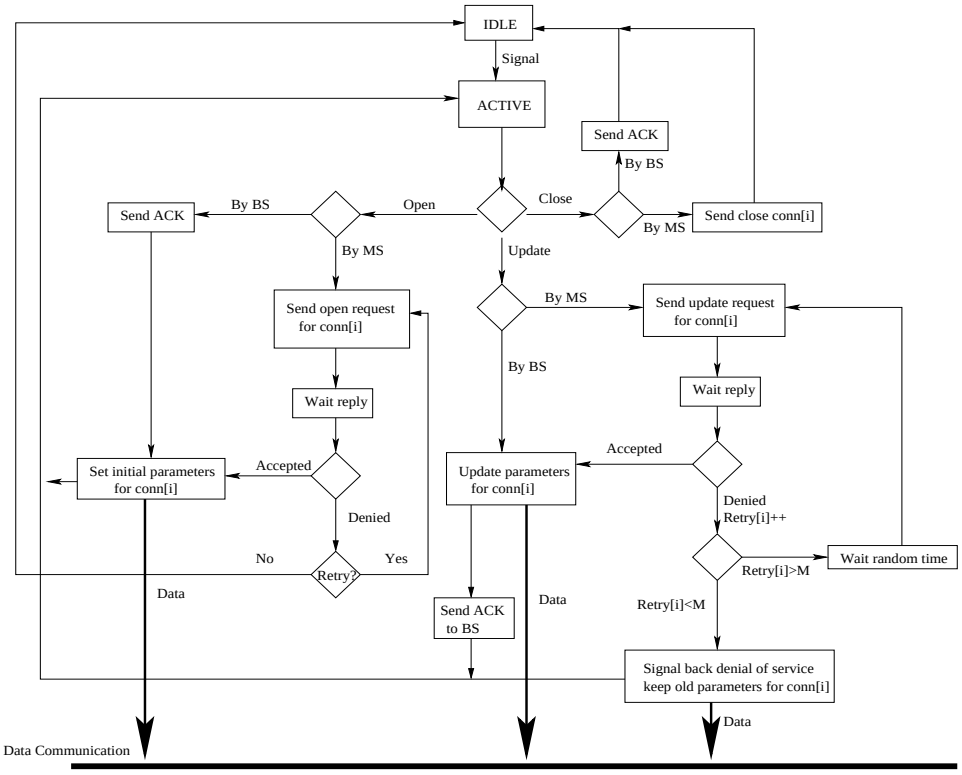

Figure 5.2: The state diagram and protocol of a mobile station.

Unlike traditional CDMA systems (IS-95), mobile stations have three types of requests: OPEN a new connection, ALTER the traffic type, and CLOSE the connection. The first two request types follow the similar steps except the fact that altering the traffic type does not change the number of users. The application in mobile unit sends an OPEN request to base station through the access channel. Along with the request is the type of the traffic (and the desired minimum data rate, if required). Our protocol defines 4 traffic types: VOICE, AUDIO, VIDEO, and DATA. Multiple data rate options are available for each traffic type. IS-95 does not allow mobile to specify the desired traffic type.

To establish a connection to the base station, the mobile sends an OPEN request, specifying the traffic type and data rate options. Once the base station receives that request, it first checks if the new request can be satisfied independent of other connections. The satisfaction is determined based on two things: The type of the traffic to be carried on the new connection, i.e., the BER requirement and minimum data rate, and interference by other users. The minimum spreading factor that will provide the BER satisfaction is used for the new connection. The maximum data rate allowed by that spreading factor is calculated meanwhile, which is compared to the minimum data rate required by the specific application and the first decision is made. The decision is either to continue remaining steps or to deny the request. If the request is denied, the mobile is allowed to retry after waiting a random time period.

5.1.2 Proposed Admission Protocol in a Base Station

Figure 5.3 depicts the state diagram (i.e., protocol) for the base stations. If it is determined that the new connection can be satisfied, the base station moves further to check if existing connections can be satisfied once the new connection is up. This facility is not available in IS-95 CDMA. However, it is added to our protocol to ensure quality for existing users. For each existing connection, the system calculates the expected average BER corresponding to the increased system load . If the expected average BER is high, we try to increase the spreading factor and check the maximum data rate in order to see if data rate requirement can also be satisfied with an increased spreading factor. This step is repeated till all existing connections are reviewed.

There are two possible situations after the review: First, no existing connection requires an update. The destination mobile is immediately informed of an OPEN request and an ACK is sent back to the requesting mobile. The second situation is that some connections may require an update. For each traffic type requiring an

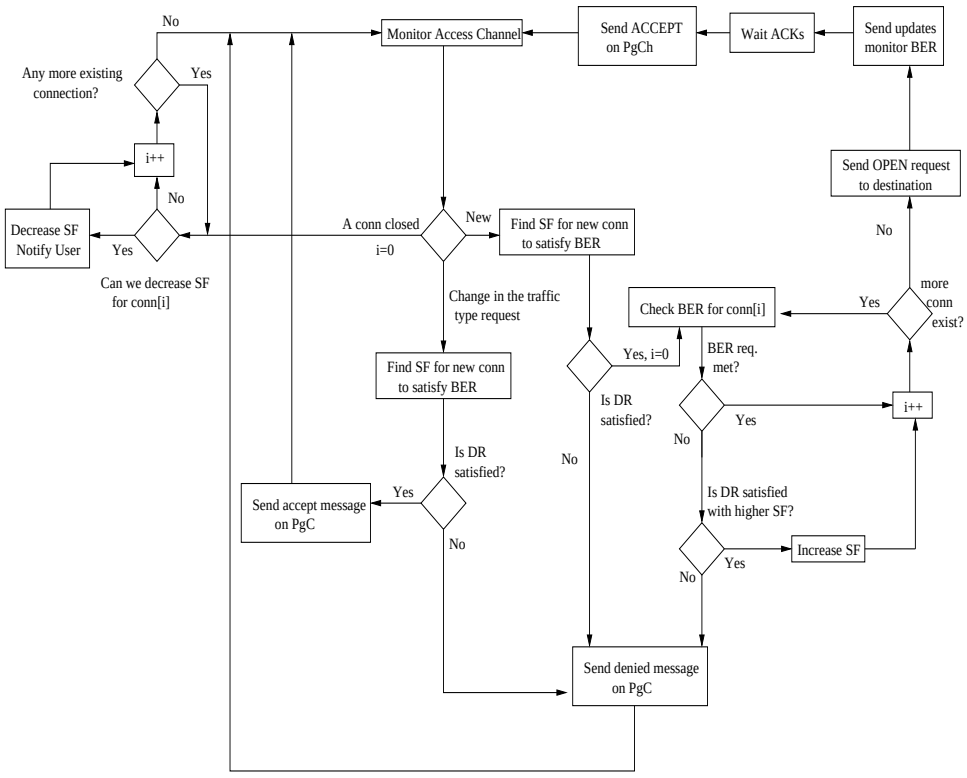

Figure 5.3: The state diagram and protocol of a base station.

update, the base station broadcasts an UPDATE message. All mobiles using the same traffic type must send an ACK back to base station to confirm that they have updated their parameters. This is necessary to make sure that no one suffers bad performance once the new connection is active. The update procedure is a major improvement over IS-95. It is the key in providing the QoS guarantee to all calls. This procedure allows the mobiles to adjust their spreading factors dynamically. When all ACKs are received, the base station sends an ACK to the requesting mobile, and the mobile may start to transmit its data.

Another important functionality of the protocol is its flexibility to alter the traffic type. One cannot switch from one traffic to another in IS-95 without terminating and re-establishing the connection. This function is provided only if a connection is already established. If the mobile decides to change the traffic type, it will send an UPDATE message to the base station. The message has to specify the requested

new traffic type. The base station will follow the same steps as in the case of the OPEN request, but this time it will not consider an increase in the number of users while doing its computations.

5.1.3 A Performance-Guaranteed System

We performed an emulation of voice traffic up to 50 users for measuring the performance of our admission control protocol. Practical parameters associated with the hardware and software environment in the mobile and base stations are listed in the following Table 5.1.

Table 5.1: Practical parameters used in the wireless WCDMA environment

Chip Rate	4.096 Mcps
Packet Size	128 bits
Spreading Factors	32,64,96,128,160
Paging Channel Data Rate	32 kbps
Access Channel Data Rate	16 kbps
BER for voice	10^{-2}
Min Data Rate for voice	8 kbps

The spreading factors used in the experiments are 32, 64, 96, 128, and 160. The experiments presented in this section only tested homogeneous voice traffic. The integration of multimedia traffic (e.g., additional support of e-mail, ftp and audio/music streams) will be presented in the next section. The BER requirement for voice traffic is assumed to be 10^{-2}. The minimum data rate required by voice is 8-Kbps[1]. For simplicity, packet size is fixed to 128 bits.

The system chip rate is set to 4.096 mcps as proposed in WCDMA standard. The maximum traffic channel data rate in this case will be 128 kbps with a spreading factor of 32. The data rates of Paging and Access channels are 32 Kbps and 16 Kbps

[1]Our latest study indicated that this is possible with advanced compression schemes such like GSM.

respectively. BER was measured in every mobile station, and the average BER among all the voice streams were calculated and illustrated in the following Figure 5.4.

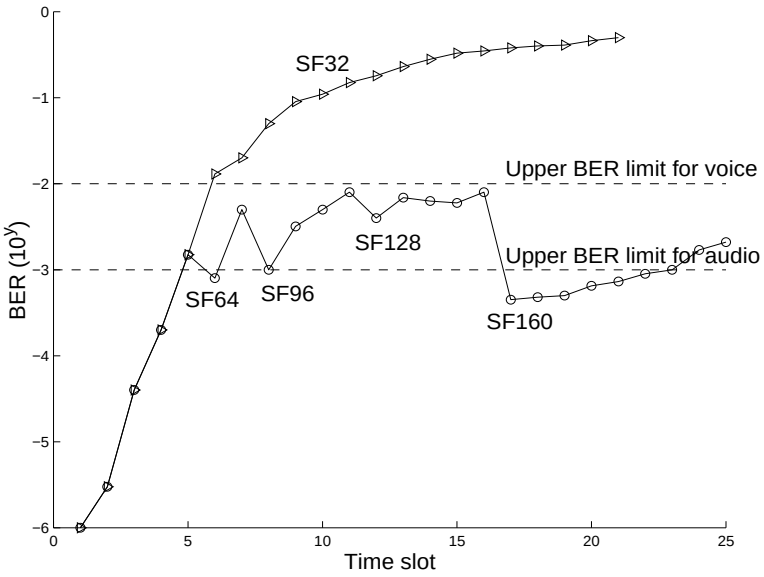

Figure 5.4: Performance guarantee with the proposed admission control.

As depicted by the curve labeled as "SF32," fixed-spreading-factor schemes did not guarantee the overall performance among all the users. When the connections are less than five, the average BER was acceptable (i.e., less than 10^{-2}). However, when the connection number exceeded five, every connection (including existing and the newly-accepted connections) suffered with the BER quality above 10^{-2}.

On the other hand, by using our proposed admission control protocol, the average BER was always maintained below 10^{-2} threshold for voice even as the number of user increases. The curve labeled with a sequence of (SF64, SF96, SF128, SF160) in Figure 5.4 stated the time instances that our CDMA system re-acted to the increasing demand of users, and new spreading factors have been adopted. The system never exceeded the upper limit for the voice traffic, (10^{-2}).

Though the experimental results are very promising, our proposed admission control protocol does introduce few consequences in two design tradeoffs. One of the tradeoffs occurs in the prolonged base station processing. The other tradeoff occurs

in the contention time for all mobile station to acknowledge the accomplishments of changing spreading factors. These two timing factors thus result in a longer end-to-end connection setup time. During the next subsections, we describe these trade-offs in details and propose methods for further improvement.

5.1.4 Processing Time at Base Station

The end-to-end delay is defined as the total time between a user making a connection request and the user becoming ready to transmit data. Two dominant time components are T_p, base station processing time, and T_{update}, waiting time for UPDATE ACKs from mobile stations.

In normal situation, to process a new connection request, the base station need to review the link status for each existing connection. Thus T_p is roughly proportional to the number of users in the system. The admission protocol takes approximately 1 msec to review one connection if it is determined that the connection can be satisfied. However, when the need for a new spreading factor is required, the required time is much higher. For instance, when the number of existing connections reaches 5 where an UPDATE should be undertaken, the processing time T_p increases to 22 msec. We realized that the connections belonging to the same traffic type always change to the same spreading factor since they have the same BER requirement. Thus, connections with the same BER requirement (same traffic type) could use same spreading factor once for all.

Contention Time for Acknowledgment

When UPDATE decision is made at the base station, our protocol requires that, before accepting the new connection, all active mobiles should switch to the new spreading factor so that BER never increases above the acceptable point. The current system design enforces that mobiles should send acknowledgments back to base station after they change their spreading factors. Since the Access Channel of a CDMA system only has a common channel and operated in slotted ALOHA, there

will be extra delays due to possible access collisions, if more than one mobile stations need to access the common channel. The average number of slots per contention is $\frac{1}{A}$ where A is the probability that some station acquires the channel in a specific slot. When $A = 1/e$, the minimum contention is reached. This results in a large contention delay when UPDATE is needed. When the number of existing connections is 9, the end-to-end delay becomes 448.4323 msec, and T_{update} is 395.432 msec.

Many approaches can potentially decrease the contention period. For examples, the goal can be accomplished if an improved collision prevention/resolution algorithm is used (i.e., other than CSMA methods). Increasing the number of access channels is another alternative. Without any technology preference, we have first investigated the second approach by increasing the number of access channels.

The base station now should distribute access channels among all users equally, thus the overall T_{update} can be reduced evenly across all the mobile stations. A hash function implemented in the base station should be sufficient for this purpose. By taking a mobile station's serial number or PN, we can balance the number of mobiles using each access channel. The following Figure 5.5 depicts the preliminary results corresponding to 1, 2, and 4 access channels.

The preliminary results demonstrate a very promising results on reducing the T_{update} component. By using 2 access channels, the average T_{update} can be reduced by 49% stably independent of the number of existing connections. With 4 access channels, a further reduction about 48% to 58% of T_{update} is accomplished.

However, when multiple access channels are deployed, we observed that the average BER will increase at the updating points. This is caused by the additional interference caused by multiple active access channels, The BER simulation with 1, 2, and 4 access channels shows that the BER corresponding 2 and 4 access channel is higher than the cases with one access channel. For two access channel, the overall BER performance is still below 10^{-2} guaranteed quality. Therefore, using two access

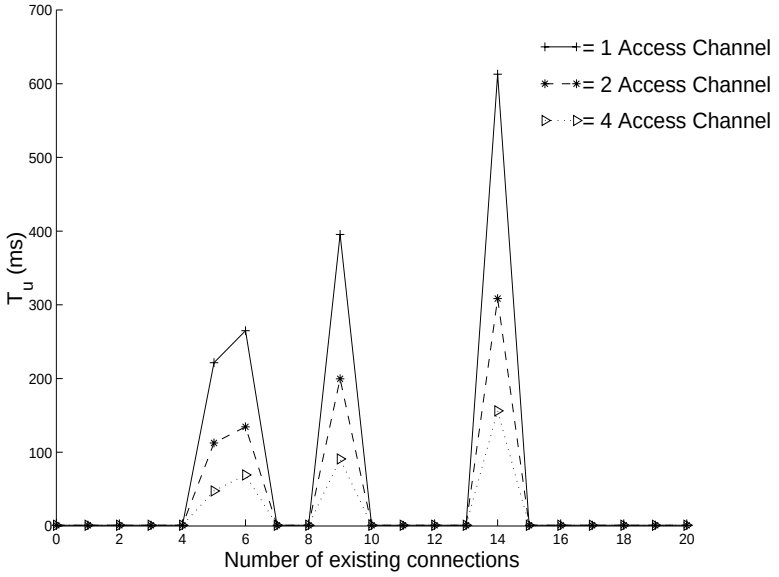

Figure 5.5: Improved T_{update} by using multiple access channels.

channels proved to be a good method to balance the shorter T_{update} time and BER quality guarantee. When we use 4 access channels, the system experiences a sharp increase in BER and violated the BER requirement at 5 users.

5.2 Dynamic Scheduling for Multimedia Integration

With a solid understanding of using dynamic spreading factors to support many voice-only users with guaranteed quality, a fundamental question needs to be answered is how can the system integrate the multimedia traffic (with the best system performance)? In this section, we will illustrate how a dynamic scheduling algorithm can be proposed for supporting this mission. To simplify the discussion, our example assumes a simple traffic pattern mixing with e-mails and voice streams. A typical traffic pattern consist of K voice sessions as background traffic, and 8 large Email requests (with a minimum BER=0.00007). We usually emulate each e-mail as 384-Kbits data amount come with a few attachments.

5.2.1 Design Issue of Traffic Scheduling for Wireless CDMA Uplink Channel

The traffic scheduling problem in CDMA uplink brings some new issues that are not presented in scheduling with wireline network. Speaking briefly, the capacity of wireless CDMA uplink is a variable of the traffic type and their spreading factors, thus makes the scheduling more difficult than the wireline network case. Let us consider a simple scheduling goal to maximize the instance throughput without any fairness requirement among different flows. Such a goal can be easily meet in time division based wireline networks by simply transmitting whatever is in the transmission buffer. As far as the transmission buffer is kept non-empty, the network should be under full utilization, which is the link speed. A often forgotten assumption behind this scheduling method is the high fidelity of communication media with very low transmitting error. Nevertheless, this assumption remain true for most of the wireline media. With a very low transmission error, the difference among traffic type in term of BER requirement is basically invisible to the network. Thus as far as the above simple scheduling goal is concerned, it does not matter as to which traffic type should be scheduled earlier.

Unfortunately, the transmission error in wireless CDMA uplink is much higher than any wireline media used today. With a given channel situation and spreading factor, some traffic might not able to be transmit due to the violation of the BER requirements. Furthermore, CDMA allow several concurrent transmissions to be undertaken, thus the throughput is the summation of data rate for all active channels. From the discussion in the first part of this chapter, it is generally true that when high spreading factors are used, the number of concurrent active channels increases. However the data rate for a channel will decrease. Thus even considering one traffic type, the decision of the spreading factor is not trivial.

In fact, the traffic scheduling in wireless CDMA uplink must decides two things: the order in which connections will be activated, and the spreading factor

to be used for each connections. The decision of these scheduling parameters should also satisfy the BER requirement for all traffic, and maximize the overall throughput. Notice that frame by frame scheduling is not considered here due to the signaling overhead between base station and mobile stations.

Therefore our simple traffic scheduler will be invoked when one of the following events occurs: new connection request, connection termination and traffic profile update. Each connection is associated with traffic type as discussed early, and the amount of data to be transmitted. For streaming traffic, a default of 500 kbits is assumed for scheduling purpose. When this quota is finished and the connection is still alive, another 500 kbits will be assigned. For non-streaming traffic such as ftp and email requests, the transmission quota is the actual amount of data for the request.

For the rest of this chapter, we will discuss the performance of such traffic scheduler with fixed spreading factor and dynamic scheduling factor. The performance is evaluated in the term of overall turn around time instead of the instance throughput.

5.2.2 Traffic Scheduling with Fixed Spreading Factor

The performance metric to be measured for data communication is the total turn-around time to fulfill all the data traffics, denoted as T_s. The goal of system design is thus targeted for shorter T_s (while maintaining performance guarantee for all existing voice connections). If traditional CDMA system with fixed spreading factor (e.g., SF=64) is used, data traffic will be experiencing the same BER as voice traffic. Therefore, when the number of background voice K increases, it may not be effective to transmit e-mails since the BER becomes too high. Since accuracy of e-mails needs to be guaranteed, the transmission of e-mail's data traffic under this situation will cause high probability of packet damage (due to the large BER over acceptable limits).

The high packets damage rate will require much higher retransmissions rate from the upper network layer. Thus the total T_s will become even higher. If the network load sustains for a long time, even the constant retransmissions will not guarantee a definite success of packet delivery. Therefore it does not only introduce a long delay for the retransmitting user, it also generates a negative interference to other users.

Thus, in wireless CDMA environment, perhaps a non-retransmission policy can benefit the overall system where data transmission occurs only under acceptable BER level. For example, when $K = 10$, the predicted average BER is 0.0001, which is higher than the BER requirement of e-mail traffic, thus email request has to wait until the voice load decreases. Based on this policy and a fixed spreading factors, a possible traffic scheduler to support integrated multimedia communication may work as follow:

Algorithm $FSF\_Scheduling$: This algorithm delays the transmission of e-mail communication until the BER in the environment is acceptable. This algorithm will reduce the frequency of the re-transmission from the upper layers.

Algorithm FSF_Scheduling

contact the base station for the current traffic load.

select an unfinished e-mail request $r(i)$**,**
check to see if the addition of this request will still satisfy BER requirement.

IF (the predicted BER exceeds any of the existing connection, or the BER of $r(i)$**),**
$r(i)$ **is not accepted for the next time frame.**
END IF

GOTO (3) and check for other traffics.
otherwise, $r(i)$ **is scheduled at the next time frame.**

We have conducted experiments with the FSF_Scheduling algorithm to collect the performance results. The focus of the performance is the total turn-around time, T_s, for 8 e-mail requests. Figure 5.6 shows the turn around time T_s with fixed spreading factor.

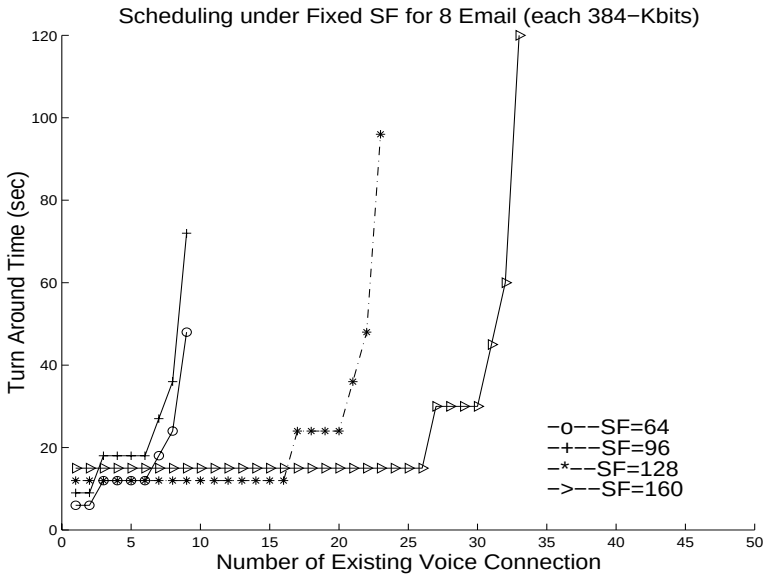

Figure 5.6: Turn around time for data traffic

The following observations can be found:

- for SF=64 , the data traffic takes 6 seconds when $k = 2, 3, 4$. It increases to 12 seconds for $k = 5$, 6 and 7. At $k = 11$, the turn-around time becomes 48 seconds. Further increase of the voice background will result in an unacceptable high BER for the e-mail traffic. Thus the transmission time is N/A. Similar performance results are observed for SF=96.

- With SF=128, T_s equals to 12 (second) when less than 16 voice connections exist. T_s increases to 24 and 36 second when background voice connections is 19 and 22. with 24 background voice connections, only one e-mail connection can be transmitted in order to satisfy the BER requirement, which results a turn around time of 96 (sec). No e-mail transmission is allowed after 25 voice users.

- For SF=160, T_s equals to 15 (sec) when less than 20 voice users presented, which is 25% higher than that of SF=128. The increase of T_s is caused by the decrease of data rate, the data rate at SF=160 is about 25-Kbits, which is 25% less that of SF=128. When $k = 27$, T_s increases to 30 (sec), 45 (sec) at $k = 31$, 60 (sec) at $k = 32$ and 120 (sec) at $k = 33$. After $k = 34$, we find that the system BER is already too high to accept any DATA traffic.

In summary, with the FSF_Scheduling algorithm, the system throughput suffers in order to satisfy the high BER requirement of DATA communication. However, with small spreading factors, the voice background can affect the transmission of DATA traffic significantly. Under light network load, the longer spreading factors (e.g., 128 and 160) actually generate longer turn-around time than the shorter spreading factors (e.g., 64 and 96). However, longer spreading factors do have the capability to support more concurrent voice streams with reasonable response time for conventional data requests. How to take advantage of these two design alternatives (with a balanced performance between continuous and conventional data service) become the focus of our next proposed scheduling algorithm.

5.2.3 Dynamic Scheduling to Improve the T_s

In order to be rescued from such undesirable situation, a scheduling algorithm based on dynamic spreading factor assignment can be adopted. Intuitively, when the current system BER does not satisfy the BER requirement of DATA traffic, we should find a longer SF. When there are multiple data transmission requests, we should select a feasible SF for each of them. Since some data sessions may last several time frames, the spreading factor should be determined in a per-time-frame basis. Therefore, minimizing the turn-around time T_s for a given set of data traffics become a global optimization problem over the all feasible SF combinations through the life-time of data sessions.

However the searching space will grow exponentially. For example, assume the data traffics consist of one FTP session and one AUDIO session , and there are 10 background voice session, we will have 3 candidate SF(SF=96,128 or 160) for AUDIO and 2 for FTP (SF=128 or 160). Further assume the AUDIO session require 256-Kbits and FTP require 64-Kbits. With the maximum channel data rate 128-Kbps (achieved if SF=32), the AUDIO session will last at least 2 seconds, which corresponding 100 time-frames (each time-frame is 20 msec). With similar

arguments, the least number of time-frame for the FTP session is 25. Thus the total combination for this example will be at least $(2*3)^{min\{100,25\}}$. In general, assume the feasible SF for k_{th} traffic type T_k is N_k, and the estimated frame number of T_k is F_k, the approximate size of searching space is $(\prod_k T_k)^{min\{N_k\}}$.

This huge searching space makes brute force searching impractical since the scheduling is desired to finish within a frame-time. In addition, the background voice traffic and data requests may change from time to time, which also makes it less worthy to find a static global optimization based on the history system load information. Therefore, it is our belief that a spreading factor combination that maximize the throughput for current time slot is more reasonable.

The remaining issue is which traffic should be considered first with what spreading factor. We observed that those with a low BER requirement have a better chance to be satisfied than those have strict BER requirement, which indicates that a small spreading factor is more likely to be accepted by the low-BER-traffic. Therefore, to improve the overall throughput, we should inspect the low-BER-traffic first. For example, with 40 voice background, we can either schedule 5 FTP traffics with SF=160, or 5 Audio traffics with SF=96, resulting in a total of 45 active traffics (including voice)[2]. Obviously the latter one has higher throughput.

Thus our scheduling algorithm works in four phases corresponding to the four data traffics, from low-BER-traffic to strict-BER- traffic. Namely, the traffic types are processed in the order of Audio, Image, FTP, and Email. The following pseudo code illustrates the algorithm for processing audio traffic. The algorithms processing other traffic types are similar.

```
Algorithm DSF_Scheduling
 /* This algorithm improves the FSF_Scheduling Algorithm
by assigning different traffic types more dynamically*/
```

[2]Other combinations which result 6 or more data traffics (such as 3 FTP and 3 Audio) will result more than 46 connections. The resulted voice BER will not be satisfied.

INPUT PARAMETERS:

N_f, N_e, N_a, N_i **represents**
the number of pending FTP, e-mail,
AUDIO and Image data requests respectively.
FTP[1..N_f**]: pending FTP requests,**
FTP[k] represents the remaining data amount yet
to be transmitted.
EMAIL[1..N_e**]: pending Email REQUESTS.**
AUDIO[1..N_a**]: pending Audio requests.**
IMAGE[1..N_i**]: pending Image requests.**
RBER[4]: BER requirement of the four traffics.
BER[5][1:50]: the predicted BER given
the number of active users
and the spreading factor.

Find the minimum spreading factor SF_v **and** SF_a**:**
$BER[SF_v][k + N_a] < RBER[1])$ **and**
$(BER[SF_a][k + N_a] < RBER[4])$

IF such SF_v **and** SF_a **are found**
$\forall\ i < N_a,\ \ AUDIO[i] = SF_a$**.**
GOTO (execute)
END IF

IF SF_v **does not** $\exists$
Use $SF_v = 160$ **as**
the voice spreading factor
END IF
locate the maximum audio traffic number $\hat{N}_a$
such that $(BER[SF_v][k + \hat{N}_a] < RBER[1])$
$\hat{N}_a = \max_i \Big\{(BER[SF_v][k + \hat{N}_a] < RBER[1])\}$

Find the minimum SF_a **that satisfies**
$(BER[SF_a][k + \hat{N}_a] < RBER[4])$

Decide which subset of audio traffics will be
chosen if $\hat{N}_a < N_a$
this should based on a fair strategy so that there is
equal chance for all traffics.

execute:
FOR (each of the selected audio traffics i**)**
Calculate T_f **as the length of time frame**
Reduce their remaining
data amount $AUDIO[i] -= \ T_f * 4.096/SF_a$
Update array AUDIO[] and N_a **by**

```
            delete finished requests
      END FOR

      Continue with other traffic types.
```

The performance of this scheduling algorithm for the sample traffic pattern is demonstrated in Figure 5.7, and the following observations are made:

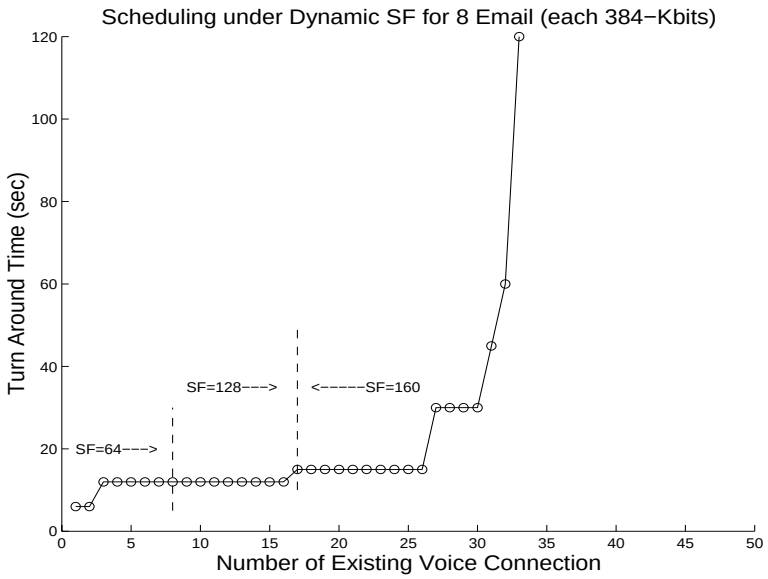

Figure 5.7: Turn around time for data traffic

- When the system is under low load (less than 3 voice), the BER of Email traffic can be satisfied with SF=64, with all Email traffics being transmitted simultaneously. The data rate is 64-Kbps. The data traffic transmission is completed within 6 (sec).

- For k between 4 to 7. The number of concurrent-transmitted E-mails are 7, 6, 5 and 4 respectively. This is because with SF=64, at most 11 active traffics are allowed to maintain the Email BER requirement. The resultant turn-around time increases to 12 (sec). Nevertheless, SF=64 still provides the shortest T_s compared to other spreading factors.

- At $k = 8$, it is observed that T_s will increase to 18 (sec) if we use SF=64. While with SF=128, we need 12 (sec). Thus the spreading factor is turned to 128 at both mobile and base stations. When k keeps increasing, the FSF_Scheduling algorithm results in a turn-around time in the range of 30 to 75 seconds (refer to Figure 5.6, which is much higher than the results from DSF_Scheduling.

- Similarly, we find that after $k = 17$, the best turn around time of 12 seconds can be approached with SF=160. In fact, after $k = 10$, FSF_Scheduling

with spreading factor of 64 or 96 can not support the data traffic with a reasonable turn-around time (due to the high packet damage probability). Our DSF_Scheduling automatically changes into high spreading factor and provide a continues transmission of data traffic. In Figure 5.7, under a heavy system load of 30 users, we still can fulfill the sample traffic pattern in 40 seconds.

- Dynamically changing spreading factor based on different system load can significantly improve system throughput. For example, with 8 background voice connections, T_s is reduced from 24 sec to 12 sec by simply turning SF from 64 to 160, which results in a 100% save in data traffic time.

In summary, our dynamic spreading factor assignment scheme can adjust the system load and balance the requirement of different traffic. Voice communication is guaranteed to be non-interrupted, and data traffics are served with reasonable fairness and response time. Our scheme can guarantee no starvation for data traffics , even when the base station is under high load.

5.3 Summary

In this chapter, we discussed a new MAC protocol for W-CDMA. The protocol can support different traffic classes that have a variety of BER requirements. Since the BER is indirectly proportional to the number of users in a wireless communication system, we designed a protocol that integrates different traffic classes with a guaranteed quality as well as maximizes the number of users supported.

The presented protocol admits new calls only if the new connection's as well as all the existing connections' quality can be guaranteed. Our protocol uses a dynamic spreading factor scheme. In order to satisfy the BER requirement, the protocol tries to dynamically change the SF used by connections that may experience a high BER. The connections may alter their SF several times depending on the changes in the number of users in the system.

We have evaluated the BER performance and admission time of the protocol under an increasing number of Voice connections. We have also compared our protocol to a regular CDMA protocol that uses a fixed spreading factor. The results show that the DSF protocol provides significant improvement in BER satisfaction

compared to CDMA with fixed SF. Even though the number of users increase in the system, the average BER of all connections are maintained below the upper limit for their traffic type. New schemes are proposed and evaluated to reduce the admission delays. Furthermore, in order to guarantee a balanced support among different traffic types, we proposed a scheduling algorithm. We measured the response time for data traffic by our proposed system.

As futher study, we evaluate the performance of the protocol under more complex condition (i.e., mixed traffic with streaming video). We also expand the protocol to cover more than one cell (i.e., so-called soft handoff). These studies are described in next chapter.

CHAPTER 6
SOFT HANDOFF WITH DYNAMIC SPREADING

In our previous study [38], we had proposed a dynamic spreading scheme such that the mobile user can change the length of the assigned spreading code for more efficient utilization of the radio frequency. With the new dynamic spreading capability, the system can support BER-sensitive multimedia traffic (i.e., combined voice and traditional text information) even when the system load is high.

It is the task of this chapter to extend the dynamic spreading scheme to the multi-cell scenario. More specifically, we need to take mobility and handover[1] [41, 42] into consideration. In one envisioned application scenario, we can image how Tedd and Tom can benifit from this scheme when driving from Gainesville to Orlando (total distance is about 80 miles). Assume base station is placed along the high way every 4 miles, the mobile will drive across 20 cells through the trip. It can be expected that there are less mobiles in the rural area than in the urban area. In rural area, Tedd can have a video phone conversation with his friend in Orlando when there are no mobile users around. When comes into a town, where more mobile users co-existed, Tedd can switched to a regular voice conversation by using a longer spreading code (thus decrease the data rate requirement and BER requirement). After driving out of the town, Tedd can resume the video session by changing back to a short spreading factor, since shorter spreading factor is enough to provide signal quality when the number of mobile user decrease (thus less interference).

The fundermental question yet to answer is: *How to integrate Soft Handoff into the dynamic spreading scheme?* In the dynamic spreading enabled CDMA system,

[1] we use the terms *handover* and *handoff* interchangeable in the context of this chapter.

a mobile might travel through a series of cells during a call session. It is likely that these cells have different number of active in-cell connections, therefore might assign different spreading factors, in order to maintain the BER quality. Thus a mobile might have to change its spreading factor from time to time as it enters into a new cell. A basic requirement for handoff algorithm under this scenario is to make sure that the updating of the spreading factor in both the handovering mobile and the target base station be finished within the allowed handoff time.

We proposed a novel soft-handoff scheme to address the above problems. Our scheme uses a similar methodology as WCDMA to determine the addition and dropping of cells into the active set. The basic functionality of the proposed handoff scheme has three parts: (1) collect information of the surrounding cells and the mobile profile, (2) update the Spreading Factor (SF) in the active-set (base stations in range) to maintain the BER level for the mobiles in these cells, and (3) decide the right spreading factor and power level for the handover mobiles.

The probabilities of updates as the result of handoff is an important factor which describes the performance of our proposed framework. We present an analytical evaluation for update probability. The preliminary results show that mobiles with voice traffic have a relative small update probability (also see Wang et al. [39]). However video traffic and WWW traffic have a high update probability, due to the high BER requirements.

The time components of the proposed algorithm were analyzed. We found that the handoff algorithm could consume a significant system resource and result in long handoff delay, especially when the system needs to execute a large number of updates. We therefore revised the handoff procedure such that handovering mobiles can be processed in batch, and several update operations caused by consecutive handoff could be combined together. Numerical results show that the average handoff delay can be reduced by almost 29%.

The core process executed in our proposed soft-handoff with dynamic spreading scheme is to determine the right spreading factor and the power level of the transmitting signal for the handoff mobile. The decision for SF and power level should be made such that the following conditions be satisfied: (1) the link quality of the mobile stations in the neighboring cell is not sacrificed, (2) the minimum BER requirements of the handovering mobile is not violated, and (3) the maximum data rate of the handovering mobile is achieved.

We proposed a sub-optimal method (called HYB) with the goal of joint optimization for SF and power level for all mobiles in the handover area. The HYB algorithm modeled the BER constraints of all mobiles as the function of various factors, including (SF, power) pair for handoff mobile, system load of the active set, and distance from the active-cells. The decision variables are *(SF, power)* for all handoff mobiles maximizing the throughput for handoff mobiles. Our modeling results in an nonlinear programming problem. Further investigation shows that our problem can be simplified to a linear programming problem with a slightly modification of the constraints. The reduced problem contains only half the number of decision variables and can be solved efficiently using linear programming methods. The solution of the reduced problem is then used to drive the desired SF's from the original constraints. Numerical experiments show that the HYB algorithm outperforms a greedy strategy (we called it MaD algorithm) significantly. The BER of the mobiles is preserved during the whole handoff period, meanwhile the interference to the surrounding cell is controlled. The throughput of the HYB scheme for WWW traffic is 25% higher compared to the conservative strategy. A 26% increase for video traffic was also observed.

6.1 Related Study

Traditional handoff algorithm is based on the voice-only network, carrying homogeneous traffic. The major performance focus of these algorithms is to guarantee

the connectivity of live calls and to keep the failure rate as small as possible. Therefore some important metrics and criteria used include received signal strength (RSS), signal to interference ratio (SIR), distance, traffic load and etc. With different handoff decision schemes, the corresponding handoff probability could be analyzed [43]. A general survey of handoff protocol was presented in Marichamy and Maskara [44].

The CDMA system supports a unique handoff scheme — soft handoff [41, 42]. Soft handoff allows more than one base station communicating with the mobile station during the handover process, thus there is no break period when the mobile moves to another cell. It had been shown that soft-handoff can reduce the transmission power of the mobile, and increase the system capacity [41, 45].

Due to the shortage of radio resources, it is essential to make the best use of the system resource and support multiple users and multimedia traffic. Resource allocation for the CDMA system and CAC in general had been discussed in many works [46–48]. In Naghshineh and Schwartz [47], a distributed call admission scheme with the consideration of neighboring cells is reported. The movement of mobiles as well as the system load in nearby cells is used to make the admission decision.

In Zhou et al. [46], a forward link power control scheme was discussed in a two-cell CDMA network supporting different QoS requirements. The scheme is optimized in the sense of minimizing base station transmitting power, and maximizing system throughput. However, the study is based on the assumption of fixed spreading factor, and did not address the resource assignment for the reverse link.

Lee and Wang [48] presented an optimum power assignment for mobiles supporting different QoS requirements. They formulated the admission problem by a set of inequality of desired SIR for all the active mobiles and provided a method to drive a feasible solution. However in this study, the processing gain is fixed to a predefined value for each traffic type, thus the admissible region will be strictly limited.

However only a few works had been done to optimize the performance of handoff mobiles by reallocating the radio resources. Furthermore, the effects of dynamically changing the spreading factor have not been reported by these studies. Our study is the first of the few attempts taking the dynamic spreading factor into the handoff algorithm. We also addressed the resource allocation during the handoff period as a joint optimization for both the spreading factor and the mobile power control. To the best of our knowledge, this approach has not been reported in literature.

The rest of this chapter is organized as follows, Section 6.2 discusses our proposed dynamic spreading soft handoff scheme and the system response time. Section 6.3 presents a baseline algorithm using a greedy strategy in deciding the spreading factor. In Section 6.4, we show how an closer estimation of the mobile/cell condition can provide high throughput while preserving the low interference level.

6.2 The Framework of Soft Handoff Algorithm with Dynamic Spreading

Our proposed handoff algorithm is based on the traditional soft handoff algorithm. The mobile station, when moving around in the multicell environment, could be either in handoff state or non-handoff state. When in the non-handoff state, the mobile is solely controlled by the resident cell, which is described in Wang et al. [38]. The mobile entered into handoff state when the active-set of this particular mobile contain more than one cell site. The maintenance of active set is similar as the one used in IS-95, where the relative pilot signals of surrounding cells are reported by the mobile and a decision of adding/dropping/ignoring of cell into/from the active set is conducted.

Figure 6.1 shows the state diagram of the handoff procedure that is executed in the BS side. The logic of the mobile station is relatively simple, and is not further elaborated here. Notice that the B_n need to send its forward Walsh code to the mobile so the mobile can decode the signal from the new base station.

It can be observed that there are three parties participating during the handoff period. The role of the mobile station in the handoff processing is similar to that of the existing soft handoff procedure. The mobile is responsible for pilot signal measurement, and compares the detected pilot signals to an adding threshold and a dropping threshold (the signal is filtered to eliminate the random variation caused by fast fading). The mobile then reports any change of signal strength to the BS in a fixed time interval. The current serving base station, will update the active set of the mobile and decide whether or not the mobile should be in the handoff period. Once in the handoff state, the current BS will gather all necessary information and decide a spreading factor for the handovering mobile.

However the handoff algorithm should also guarantee that the target cell (and other nearby cells) can still operate with performance guarantee. In fact, the handovering mobile station is usually relatively close to the BSs in the active set, so their signal is strong enough to become dominant interference if not being carefully controlled. Thus it is the duty of the handoff algorithm to maintain the desired BER in the target cell all the time, e.g., an update in the target cell should be taken immediately. The need of updating the spreading factors in target cell during the handoff procedure could prolong the total handoff time. This makes the time factor more important than in the traditional system, where handoff can be finished within a matter of 100 ms. In the case of heavy handoff traffic, efficient handoff processing becomes nontrivial since the target cell might have to trigger update procedures several times in a limited time period.

The performance goals of the handoff algorithm with dynamic spreading factor are summarized as follows (1) All handoff requests should be processed within a given time period, such that the mobile can smoothly migrate to the other cell. (2) The algorithm should be able to handle stress handoff requests, which is important when the system in under heavy load. (3) The handoff algorithm should seek the highest

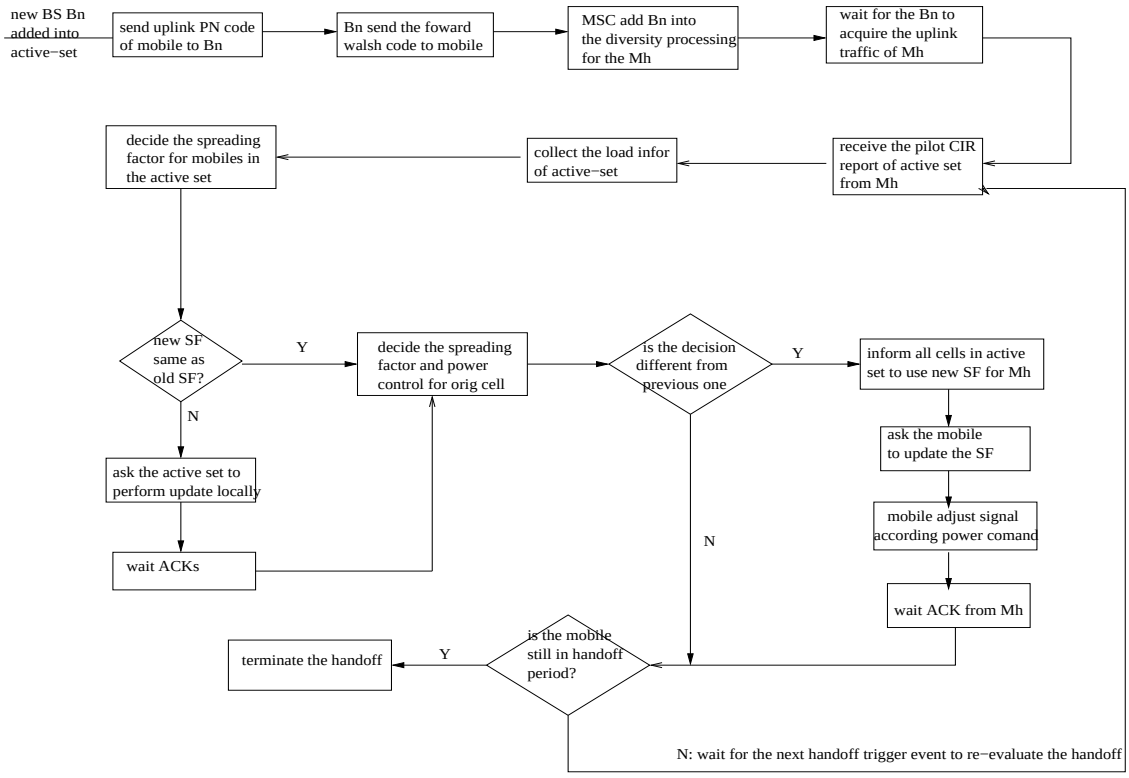

Figure 6.1: Framework of soft handoff scheme at BS with dynamic spreading

throughput while maintaining the BER performance of the handoff mobile during the handoff period. The performance of the active cells must not be jeopardized. The BER performance of these cells also can't be violated at anytime.

6.2.1 Stationary System Behavior

Basically, the handoff algorithm need to decide: (1) whether or not the relevant active-set should perform an update process, and (2) whether or not the handovering mobile should change its spreading factor. In order to study the stationary update behavior, we define the mobile state by the spreading factor it is using. The possible values of the spreading factor is limited to $\{16 * k | k = 1, 2, ..., 8\}$. We will study the updating probability for the handoff mobiles using a state transition diagram, as shown in Figure 6.2.

For a mobile to transit to a new state, certain transition conditions must be satisfied. Specifically, a transition occurs when the target SF is different from the

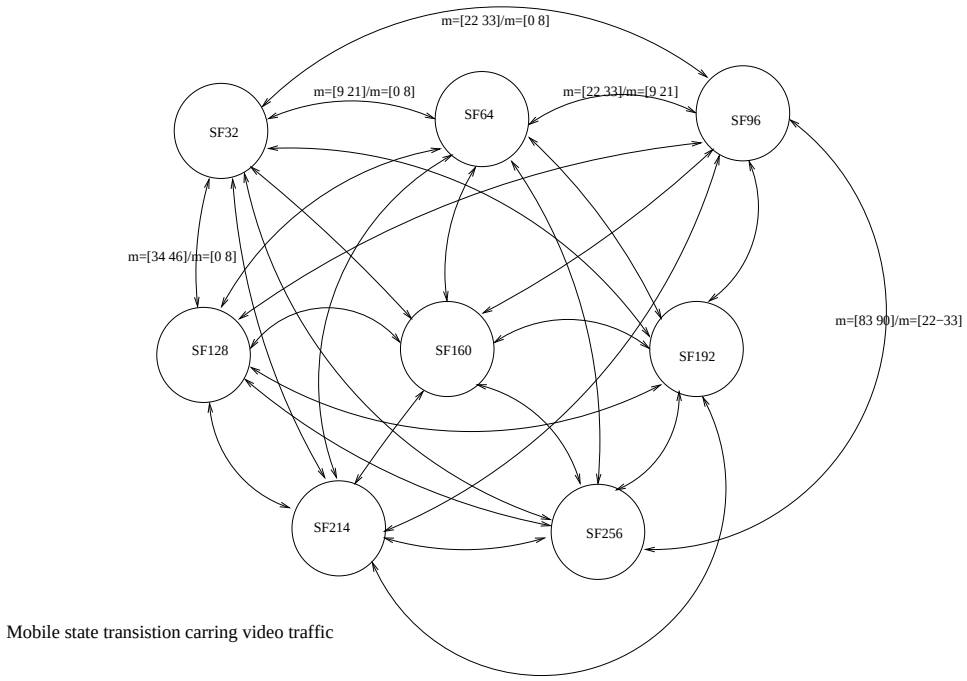

Figure 6.2: State Transition For Handoff Mobile

current one. Thus the the driving events of the state transition is the change in the number of active mobiles in the target cell. We use a double-arrow to represent the forward and backward transitions between two states. For the sake of convenience, the two states associated to each transition is called low state and high state, with low state has a shorter spreading factor. The transition is labeled by a trigger condition pair *(x/y)*, where x represent the transit condition from the low state to high state, and y for the reverse direction. The transition condition can be derived from the admission criteria for a particular traffic type. For instance, we can use Figure 4 in Wang et al. [38] to obtain the situation when an updating of SF will occur. Due to the complexity of the graph, we only labeled a few transitions in Figure 6.2.

Let $Prob_i$ represent the probability for mobile in state i. The probability of update for the handoff mobile can be expressed by:

$$Prob^v_{u,m} = \sum Prob^v_i \sum_{j \neq i} Prob^v_j = \sum Prob^v_i (1 - Prob^v_i) = 1 - \sum (Prob^v_i)^2$$

Here the subscripts u, m indicate the update in the mobile unit, and the superscript v represents voice traffic. We will use subscripts u, c to represent the update in the target cell.

$Prob^{v}_{u,m}$ can now be calculated from the probability for each state of the voice traffic. Assume the system load in the target cell is uniformly distributed in the admissible range, we can use the frequency of the i^{th} SF as the approximation of $Prob_i$. The relationship between the SF and the system load for different traffic types can be derived from the BER-SF-SystemLoad curve as shown in Wang et al. [38]. Following discussion assumes that a particular correspondence between SF and system load is already obtained. For example, assume the maximum voice connection is fixed to 87, the SF update policy will use **SF=32** for voice traffic when the system load is from 11 to 20. This indicates $\frac{(20-11)}{87}$ of the 87 possible cell load situation result in **SF32**. Similarly, the frequencies for other mobile state can be obtained. The approximated probability for spreading factor **(SF16, SF32, SF64, SF96, SF128, SF160, SF192, SF224, SF256)** are **(0.115, 0.126, 0.23, 0.068, 0.046, 0, 0, 0, 0)** respectively. The corresponding probability of update for voice mobile is thus $Prob^{v}_{u,m} = 1 - .115^2 - .126^2 - .23^2 - .068^2 - .046^2 = 0.702$. We can drive the probabilities of update for video and WWW traffic similarly. The probabilities of update for video and WWW traffic are $Prob^{www}_{u,m} = 0.82$ and $Prob^{video}_{u,m} = 0.87$.

The probability of update in the target cell can be decided from the update policy too. A cell load m is referred as an unstable point if one of the traffic types needs to update to a new SF when the system load become $m + 1$. We found that there are 18 unstable points among the 87 possible cell load, thus the probability of update in target cell is ($Prob_{u,c} = \frac{4+7+7}{87}$) 0.21.

Now we can derive the probabilities for the four situations based on the updating decisions for the mobile and target cell: 1) **(unchanged, unchanged)**, 2)**(unchanged, update)**, 3) **(update, unchanged)**, and 4) **(update, update)** by the

following equations:

$$
\begin{aligned}
P1 &= Prob(\textbf{mobile update}, \textbf{cell update}) \\
&= \frac{1}{3}(Prob^{v}_{u,m} + Prob^{video}_{u,m} + Prob^{www}_{u,m})Prob_{u,c} && (6.1) \\
&= 16.6\% && (6.2) \\
P2 &= Prob(\textbf{mobile update}, \textbf{cell unchanged}) \\
&= \frac{1}{3}(Prob^{v}_{u,m} + Prob^{video}_{u,m} + Prob^{www}_{u,m})(1 - Prob_{u,c}) && (6.3) \\
&= 62.5\% && (6.4) \\
P3 &= Prob(\textbf{mobile unchanged}, \textbf{cell update}) \\
&= \frac{1}{3}((1 - Prob^{v}_{u,m}) + (1 - Prob^{video}_{u,m}) + (1 - Prob^{www}_{u,m}))Prob_{u,c} && (6.5) \\
&= 4.9\% && (6.6) \\
P4 &= Prob(\textbf{mobile unchanged}, \textbf{cell unchanged}) \\
&= \frac{1}{3}((1 - Prob^{v}_{u,m}) + (1 - Prob^{video}_{u,m}) + (1 - Prob^{www}_{u,m}))(1 - Prob_{u,c}) && (6.7) \\
&= 16\% && (6.8)
\end{aligned}
$$

Substituting the updating-probability for handoff mobile and target cell $Prob_{u,m}$ $Prob_{u,m}$, $Prob_{u,m}$ and $Probu, c$ into equation (6.2)-(6.8), we have: $P1 = \frac{1}{3}(0.702 + 0.82 + 0.87)0.21 = 0.166$, $P2 = 0.625$, $P3 = 0.049$, and $P4 = 0.16$. The above results show that 63% of the time there will be an update for the handoff mobile and unchanged in the target cell. The probability of no-updating in both sides is only 0.16. This indicates that our proposed handoff scheme could lead to considerable amount of updates if handoff occurred. In the following discussion, the handoff timing is analyzed for dynamic system behavior.

6.2.2 Handoff Time Analysis

The handoff processing time is defined as the total time between a mobile entering into handoff state and the user become ready to communicate with the active set. We can formulate this by: $T_{handoff} = T_i + T_u + T_e$

where T_i is the time to exchange information among the active set. This includes three parts: the link parameter of the handovering mobile needs to be sent to the new member in the active set T_m, the system load information of each cell should be sent to the master cell T_{load}, and a time delay for the target cell to acquire the handoff mobile T_{target}. T_m and T_{load} are mainly caused by the communication delay between the *master cell* and the other cells in the active set. We use a round trip delay of 10 ms as an reference value (this is usually observed between two hosts in the same wired network). T_{target} can be approximated by the soft handoff time in the traditional CDMA system, which includes the time for user detection, resource allocation and etc. A practical upper bound of T_{target} is in 40 ms range.

T_u is the time to evaluate the impact to the target cell caused by the handoff mobile. During this time period, it must be decided whether or not an update in the target cell should be performed. If so, the updating must be completed before proceeding further. The duration of this part depends on the number of active users in the target cell, and how many of these need to be updated[2]. As shown in Wang et al. [38], this part could be further divided into processing time to evaluate the BER for the target cell, and the updating time. Table 6.1 illustrates the increasing of T_u as the number of existing mobile users grows. Each column of the table represents a point when a change of the spreading factor is necessary compared to the previous column. For instance, assume the target cell has two video users and one WWW user already. When a new mobile arrived, it will take 9 msec for searching time, and require two update operation for the two video users (corresponding to 20 msec). Overall, we have $T_u = 29$ msec. This case corresponds to the column with 4 total users.

[2]The evaluation of BER is performed for all active users, and only part of them need to update their SF (e.g., all the users carrying video traffic, or all WWW users for next instance). Notice also that here we assume that the updating is performed in a sequential manner.

Table 6.1: Processing time in target cell at update point in msec

Total Users		2	3	4	6	8	11	15	16	19	22
Spreading Factor Used	WWW users	16	32	32	48	48	64	80	80	96	96
	Streaming Video	16	16	32	32	48	48	48	64	64	80
	Voice	16	16	16	16	16	32	32	32	32	64
Searching time		7	8	9	11	13	16	20	21	24	27
Update time		30	20	20	20	30	40	50	70	80	80
Overall T_u		37	28	29	31	43	56	70	91	104	107

Total Users		23	27	32	34	36	39	40	44	45
Spreading Factor Used	WWW users	112	128	144	144	160	160	176	192	192
	Streaming Video	80	96	96	112	112	128	128	128	144
	Voice	64	64	64	64	64	96	96	96	128
Searching time		28	32	37	39	41	45	46	50	51
Update time		90	180	110	120	120	130	130	140	140
Overall T_u		118	212	147	159	161	175	176	190	191

Finally, T_e is the time to choose the new spreading factor and signal power for the handoff mobile, and the update time of the handoff mobile if needed. This procedure needs to be re-executed after a period of time such that the spreading factor and power level of the handoff mobile can be adjacent according to the mobile location.

It is our concern that the handoff process described in Figure 6.1 will have efficiency problems, due to the fact that it processes handoff requests in a sequential manner. The numerical results of the overall handoff time corresponding to different handoff rates are plotted in Figure 6.3. At moderate handoff rate ($\lambda = 2$), the handoff time is well controlled. The base station spent about 100 ms for each handoff mobile when there was no update in the target cell. When there is update in the target cell, we observed significant increase of processing time, represented by the spurs. However, for a high handoff rate ($\lambda = 8$), the overall processing time starts to build up for the late arrivals. The slope of the peaks of the response time increase steadily after 36 users. The highest processing time (1800 ms) was required when there are 87 mobiles already in queue. To reduce the processing time of the handoff procedure,

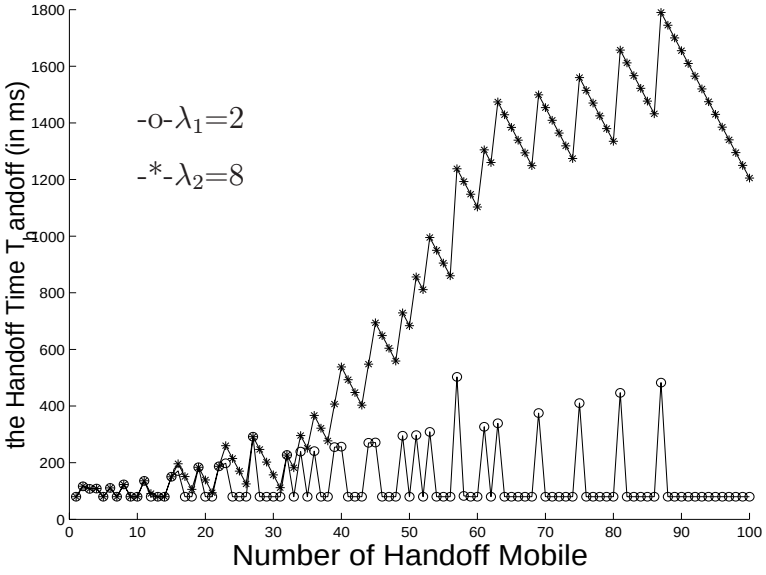

Figure 6.3: (a)Handoff time under stress attack

one immediate improvement is to allow the handoff algorithm to handle multiple handoff requests each time. Our proposed batching algorithm is invoked by a timer, or the event of new handoff requests, whichever comes first. The revised handoff algorithm is shown below:

```
Procedure HandoffProcess
Global Varibles:
        hr[]; /*handoff requests queue*/
        nhr[]; /* new handoff request queue */
        act_union; /*the union of active set
        act_set[i]; /*active set of i-th handoff mobile*/

while (1){
        if (nhr[] is empty) then
                sleep (10 ms);
                goto oldqueue;
        else
                m= number of handoff request in nhr[];
        end
/* gethering the system status of active cells for all handoff
requests */
                for (i=0;i<m;i++)
                 act_union= union (active_set(i), act_union);
                end
                for (each cell in the act_union)
```

```
                receive_cell_stat();
              end
/* re-evaluate the BER conditions for the cells in act_union,
decide wether an update is needed*/
              for (each of i-th cell in the act_union)
                load = act_union[i].load + handoffnumber[i];
                for (each traffic type ty[j])
                   SF[i][j] = cal the desired spreading factor;
                   if SF[i][j] != the previous spreading factor
                      update[i] = true;
                end
              end

              send the new SF[i][j] to the active_union
                 whose update[i] == true;
              wait ACK from all updating cells;

/* evaluate the BER and Spreading factor for the handover mobile
   considered in this batch */

oldqueue:
              for (each of the hr[k] in this batch)
                hr[k].master = decide the current master cell ;
                hr[k].SF = calculate the desired SF;
                hr[k].pw = calculate the desired power level;
              end
              execute the decision for hr[]'s
              wait for ACK from hr[];
          /* each mobile will send ACK to its master cell */
}//end while
```

One design parameter of the batching scheme is how to choose the right value for time-out for the timer, or batching period. A small batching period might not help much in case of heavy handoff traffic. On the other hand, a large batching period will increase the handoff delay in general. Due to the space limitation, we will discuss this issue in our future publications. The batching period used here is set to 10 ms, which is the same frequency of SNR reported from the mobile stations.

The performance of the improved scheme is plotted in Figure 6.4. The system response time for the small handoff traffic is not changed. At $\lambda = 8$, the processing time at update points reduced significantly, mostly concentrated between 600 to 800

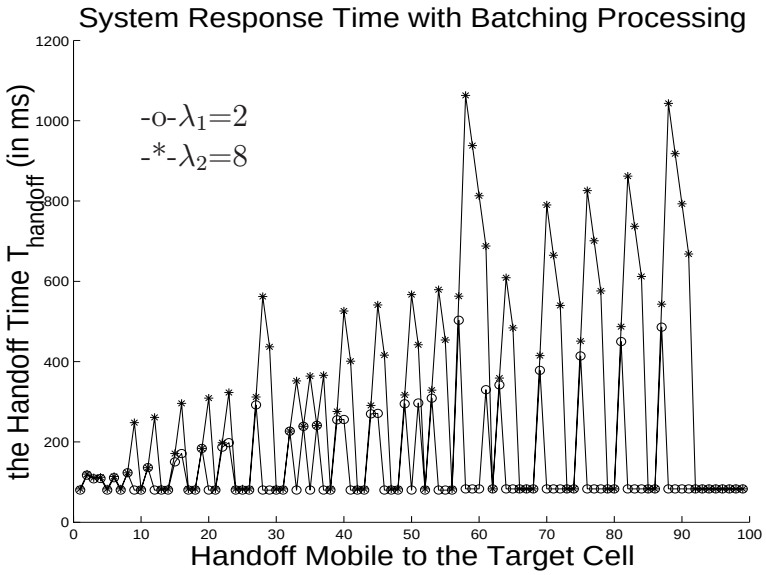

Figure 6.4: Improved handoff time under stress attack

ms, The maximum BER is 1090 ms, which is 60% of the original one. Furthermore, the build up of the waiting time is eliminated.

6.3 Determine Spreading Factor for Handoff Mobile

6.3.1 Design Factor

For the mobiles in the handoff area, the system should decide two critical parameters dynamically—spreading factor and transmitting signal strength. Since the determination of the spreading factor should be based on the BER performance, we need to consider all the different radio links among the mobile and the active set.

We first consider a simple decision scheme where the decision of the SF and the power level is separated. The SF is decided using a greedy method, called Maximize-Data-rate (**MaD**) scheme, which always choose the shortest possible SF that can be supported in the active cells. The **MaD** can be implemented by performing the admission control algorithm in the single cell situation for each of the active cell. However, this could result in a degradation of the link quality. If the base station advertising the short SF code is under deep fading, the reverse link will suffer in great deal, since the other BSs presumely cannot guarantee the desired BER with

a short SF. Another extreme case takes the opposite choice: it always selects the longest spreading factor. This certainly can guarantee BER quality in most of cases, however, the radio efficiency is decreased in this rather conservative scheme.

We examine the performance of the **MaD scheme** coupled with two different power controls: (1) the traditional **OR-ON-DOWN** algorithm and (2) follow the power control command of the *master cell*. We find that none of the two examined power control methods can provide good performance in all cases. The former one fails the BER requirements of handoff mobile, and the later one causes too much interference to other cells. The problem is solved in Section 6.4, where we propose an optimized approach based on a more precise estimation of BER, where the SF and power level of mobile is driven from a constrained inequality array specifying our BER model in the handoff area. The simulation results show that the optimal scheme can provide the good throughput while satisfying necessary BER requirements for the other parties.

6.3.2 Performance of OR-ON-DOWN Power Control

The OR-ON-DOWN scheme is a close-loop power control used in IS-95. The base stations in the control set can issue a power control command to the mobile based on the received signal strength. The power command is a 1-bit information embedded in the forward channel every 1.25 ms. An 'UP' command means that the mobile should increase its transmission power by a fixed step, and a 'DOWN' command means to do the opposite.

As described in numerous research literature, this power control method requires only one vote to decrease the mobile transmission power, while all positive votes are required to increase. This method is proven successful for the voice only traffic where a fixed SF scheme is used. However, with the dynamic spreading factor enabled system, the mobile might not be able to communicate with all cells in the active set well, particularly with the high loaded cell. For instance, assume a mobile

handoffs from a *cell*0 with load of 10 (desired SF=32) towards *cell*1 with load of 20 (desired SF=64), the SF assignment algorithm described above will take *cell*0 as the main path, thus choose a short spreading factor of 32. Now if *cell*1 issues a 'DOWN' command to the mobile (since the mobile is moving towards *cell*1 and finds that the signal is becoming too strong), the mobile will follow the command as required by the 'OR-ON-DOWN' rule. This will in turn reduce the received signal strength in *cell*0, thus may increase the link BER. For the reverse link between the handovering mobile and *cell*1, the BER is also high since the assigned spreading factor of 64 is only good for *cell*0, and not long enough in the heavy loaded *cell*1. Therefore the overall link quality will be below the expected since neither link can provide good performance.

Using the above example, Figure 6.5 illustrated the BER of handoff mobile in the handoff area. The target BER is set to 2% for voice conversation. Figure 6.5.(a) shows a scenario when the mobile moving from a light loaded system (*cell*0) to a higher loaded system (*cell*1). The right figure shows the result when mobile moves in the reverse direction.

In the case of forward movement (*cell*0 $\rightarrow$ *cell*1), the mobile can maintain an acceptable BER of 0.008 in the range of 800 meter to 1000 meters. After passing the midpoint between *cell*0 and *cell*1, which is at the distance of 1000 meter, the BER increases rapidly. At around the 1040 meter point, it exceeds the 0.02 threshold and continuely increases. Then the mobile is solely controlled by *cell*1 and use a new spreading factor of 64, and the BER drops to 0.01, an acceptable level.

When the mobile moves from *cell*1 to *cell*0, we observed the opposite behavior of BER. At range 800 meter, when the mobile enters the handoff area, there is a sudden increase of BER. At that point, the spreading factor is adjusted to 32, according to our greedy algorithm. Meanwhile the OR-ON-DOWN power control will limit the signal power such that *cell*1, which is closer to the mobile, receive the

signal necessary strong. Thought the received signal level is maintained in *cell*1, the change of SF from 64 to 32 significantly increases the BER in the *cell*1. As to the *cell*0, which is received weaker than expected signal, due to the 'DOWN' power command issued to the mobile from *cell*1, also cannot provide the expected link quality. Overall, the BER in this period exceed the allowed value.

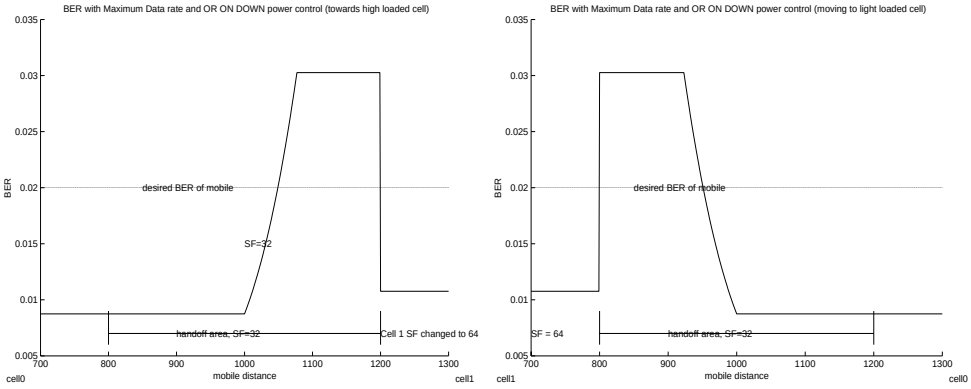

Figure 6.5: Handing off mobile's BER for MaD/OR on DOWN power control

6.3.3 Main Path Power Control

A lesson from the OR-ON-DOWN scheme indicates that at least one of the radio link should be strong enough to provide the required BER. We denote the radio channel between mobile and the cell allowing the shortest spreading factor as the *main path*. To guarantee that the *main path* is good through the handoff period, we let the cell in the *main path* carry out the power control. The same close-loop control is performed by issuing 1-bit power commands. The BER performance of this method is plotted in Figure 6.6a. We only showed the BER observations for SF=32, 64 and 92. The system setting is the same as in the example of the previous section.

The simulation result shows that the required BER for the mobile is satisfied through the handoff period. We particularly noticed that the BER decreases considerably at half of the handoff area close to *cell*1. The lowest BER of 0.00001 is observed around the right edge of the handoff region. Then the mobile change to

SF=64 and is controlled by *cell*1. The BER also rebounded back to 0.01 level. The low BER during the handoff area is caused by two factors:(1) during the handoff period, the mobile's link with *cell*0 is the *main path*, this along guarantees that the BER observed in *cell*0 is not higher than 0.008, (2) as the mobile moves away from *cell*0 (towards *cell*1), the distance from *cell*0 is increasing, which demands the increase of transmitting power in the mobile. Meanwhile, the distance to *cell*1 is reducing, together with the increasing of the transmitting signal level, the received signal at *cell*1 increases in a very rapid manner (received signal is $\left(\frac{r0}{r1}\right)^d$ times expected). The high received signal strength in *cell*1 provides a compensation for the high interference in *cell*1, and results in an even better link quality at *cell*1. To further verify

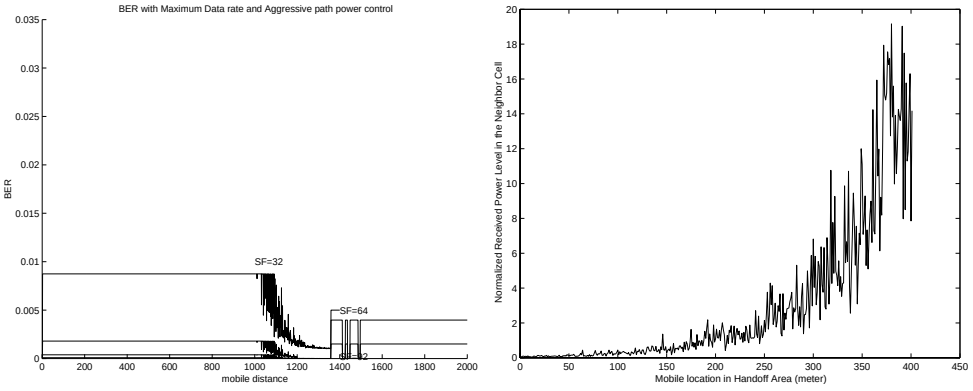

Figure 6.6: (a)Handing off mobile's BER for MaD/main-path, and (b)the received power of handing off mobile in the neighbor cell using MaD scheme

the effect of the *main path* method, we showed the strength of the received signal at *cell*1 in Figure 6.6b. As can be observed, the lobe of the signal strength increases polynomially, with random variation caused by fast fading. At the end of handoff area (around 400 meters), the received signal is 32 times higher than the normal level.

However, the strong signal of the handoff mobile also contribute to the interference level for other users in *cell*1. The BER level of the voice users (using SF=64) in *cell*1 increase rapidly as the handoff mobile's approaching the target cell. The

bottom-line BER (0.02) is violated after 1100 meter, and the highest error rate is 0.37 before the end of the handoff period.

6.4 Performance Optimization in Handoff Period

To optimize the performance in the handoff area, we proposed an open loop spreading factor and power control algorithm based on an accurate estimation of the link BER for both the handovering mobile and the cells in the active set. The algorithm explicitly calculates the desired SF and transmitting power level for all handoff mobiles. To better understand our algorithm, we shall discuss how the BER is estimated, and by which means it is related to our target parameters of SF and signal power. Our modeling of BER for the handoff mobile and active set results in a serial of inequality that imposes constraints on the possible values for SF and power level.

6.4.1 BER Model in Handoff Area

The approximate of the channel BER can be expressed as a function of $\frac{E}{I}$ (SIR, or energy-per-bit-to-interference ratio). Let $SIR_{j,i}$ be the SIR of the j^{th} handoff mobile at the i^{th} cell, we have:

$$SIR_{j,i} = \frac{E_j}{\eta_i} = \frac{p_j SF_j}{m_i + I_h + fI_o} = \frac{\left(P_j r_{j,i}^{-\mu} 10^{\frac{\eta}{10}}\right) SF_j}{(m_i + fI_o + I_h)} \tag{6.9}$$

$$I_o = \frac{1}{|A|} \sum_{u \in A} m_u \tag{6.10}$$

$$I_{h,j} = \sum_{k \neq j} \left(P_k r_{k,i}^{-\mu} 10^{\frac{\eta_k}{10}}\right) \tag{6.11}$$

here p_j is the received signal power at cell i for mobile j, and SF_j is the spreading factor of j^{th} mobile. m_u is the system load of cell i. f represent the factor for other cell [41], and A is the set of active cells. In estimating the interference from other cells (I_o), we use the average number of mobile users in the surrounding cells as the mobile density (number of mobiles/cell). However, the interference caused by the handoff mobiles I_h needs to be considered as a separate source from the general

other-cell-interference term, since they might transmit stronger signal than expected (as shown in Section 4).

The received signal strength is modeled by a path loss model: $p_{j,i} = P_j r_{j,i}^{-\mu} 10^{\frac{\eta}{10}}$, where P_j is the mobile's transmitting power, $r_{j,i}$ is the distance between the mobile and base station, μ (=3.5) is the path loss order and η is the zero mean random power fluctuation. We also assume that the non-handoff mobile is under perfect power control of its resident cell, with a normalized received power level of 1[3].

Using the diversity decoding for uplink channel, the best link among the active set is chosen for the j^{th} handoff mobile. Let A_j be the active set of j^{th} mobile, we have $SIR_j = \min_{i \in A_j} SIR_{j,i}$.

Substituting (6.10) and (6.11) into equation (6.9), the SIR for the j^{th} handoff mobile can be expressed as:

$$SIR_j = \min_{i \in A_j} \left[\frac{\left(P_j r_{j,i}^{-\mu} 10^{\frac{\eta_j}{10}} \right) SF_j}{\left(m_i + \frac{f}{|N_i|} \sum_{u \in N_i} m_u + \sum_{k \neq j} \left(P_k r_{k,i}^{-\mu} 10^{\frac{\eta_k}{10}} \right) \right)} \right] \tag{6.12}$$

The SIR for the in-cell mobiles in each active set could be similarly obtained. Note that the interference from all the handoff mobiles is now regarded as noise, and the regular in-cell mobile users have an unit received signal strength. For traffic type t in the i^{th} active cell with spreading factor SF_t, the SIR is approximated by:

$$SIR(i,t) = \frac{SF_t}{(m_i + \sum_j \left(P_j r_{j,i}^{-\mu} 10^{\frac{\eta}{10}} \right) + fI_o)} \tag{6.13}$$

With both BER requirements for the handoff mobile and in-cell mobile (equations (6.12) and (6.13)) satisfied, we should choose the SFs such that the maximum throughput is obtained. Since the data rate is inversely proportional to the SF, the overall throughput is the sum of $\frac{B_c}{SF_j}$ for all handoff mobiles. Here B_c is the chip rate of the system (5 Mcps for WCDMA). Let H denote the set of handoff mobiles at any time,

[3]In practice, the received power of different traffic are not uniformly assigned, thus they cause a different level of interference.

the optimal SF/power assignments for the handoff mobile can be obtained by solving the following problem:

$$\max z \qquad = \sum_{j \in H} \frac{B_c}{SF_j} \tag{6.14}$$

$$SIR_j \qquad \geq b_j, \qquad j \in H \tag{6.15}$$

$$SIR_{i,t} \quad \geq b_t, \qquad i \in A, \text{ and } t \in \{\text{traffic type set}\} \tag{6.16}$$

6.4.2 Simplification of the Original Problem

The nonlinear optimization problem described by (6.14) - (6.16) certainly could be solved by many well-known methods (e.g., Newton's method, or gradient algorithm). However these methods may not be efficient enough, especially when the number of variables to be optimized is large. Therefore it is our interest to find an efficient algorithm to deliver sub-optimal results.

We start with series of simplifications of the original problem, such that the problem is tractable. The first step is to eliminate the minimizing operation in (6.12). For each handoff mobile with given SF_j and P_j, the minimum SIR path is where the shortest distance $r_{j,i}$ and lowest load m_i is, according to (6.12). However, the two factors ($r_{j,i}$ and m_i) may conflict with each other, in that the closest cell might not be the one with the least number of in-cell traffic. We therefore define a variable $w_i = \frac{m_i}{r_{j,i}^{-u}}$ to determine the main path for mobile j. Thus (6.12) can be simplified by the following rule:

$$SIR_j = \frac{\left(P_j r_{j,i(j)}^{-\mu} 10^{\frac{\eta_j}{10}}\right) SF_j}{\left(m_{i(j)} + fI_o + \sum_{k \neq j} \left(P_k r_{k,i(j)}^{-\mu} 10^{\frac{\eta_k}{10}}\right)\right)} \qquad i(j) = arg\min_{k \in A_j} w_k \tag{6.17}$$

After neglecting the random fluctuations of path loss and rearranging (6.15), the SIR constraints for the handoff mobile become

$$\left(P_j r_{j,i(j)}^{-\mu}\right) SF_j \geq \left(m_{i(j)} + fI_o + \sum_{k \neq j} \left(P_k r_{k,i(j)}^{-\mu}\right)\right) b_j \tag{6.18}$$

To further simplify (6.18), it is noticed that $P_k r_{k,i(j)}^{-u}$ is nonnegative, thus

$$\left(m_{i(j)} + fI_o + \sum_{k\in H}\left(P_k r_{k,i(j)}^{-\mu}\right)\right) b_j \geq \left(m_{i(j)} + fI_o + \sum_{k\neq j}\left(P_k r_{k,i(j)}^{-\mu}\right)\right) b_j$$

Therefore, (6.18) is replaced by a strong condition

$\left(P_j r_{j,i(j)}^{-\mu}\right) SF_j \geq \left(m_{i(j)} + fI_o + \sum_{k\in H}\left(P_k r_{k,i(j)}^{-\mu}\right)\right) b_j$, Removing the random variable and rearrange the expression[4] in (6.16), The simplified problem now becomes:

$$\max z = \sum_{j\in H} \frac{B_c}{SF_j} \tag{6.19}$$

$$\left(P_j r_{j,i(j)}^{-\mu}\right) SF_j \geq \left(m_{i(j)} + fI_o + \sum_{k\in H}\left(P_k r_{k,i(j)}^{-\mu}\right)\right) b_j \tag{6.20}$$

$$\sum_{k\in H}\left(P_k r_{k,i}^{-\mu}\right) \leq \frac{SF_t}{b_t} - m_i - fI_o \qquad i \in A \tag{6.21}$$

$$SF_j, P_j \geq 0 \qquad j \in H \tag{6.22}$$

It can be easily seen that the above constraints are feasible only if $\frac{SF_t}{b_t} - m_i - fI_o > 0$ for each active cell i. Any positive power vector satisfying (6.21) can be used to drive a corresponding SF_j from (6.20), since given a positive P_j, we can select a SF_j big enough to be greater than the right hand of constraints in (6.20).

It is time now to rewrite the objective function in a linear expression. Our basic strategy is to replace the term $\frac{1}{SF_j}$ in (6.19) by a linear combination of P_j. We noticed that the first order partial derivative of the objective function is always negative($\frac{\partial z}{\partial SF_i} = -\frac{B_c}{SF_i^2}$) , which means that the optimal SF_i must be in the boundary defined by the constraints. Once the constraints and the objective function are transformed into linear format, we can use a linear programming method to solve the problem.

For the sake of convenience, we define

$$c_i \ = \frac{SF_t}{b_t} - m_i - fI_o \qquad i \in A \tag{6.23}$$

[4]By ignoring the random components as presented in the actual system model, our solution is only valid as a stationary approximation.

the constraints in (6.20) can be rewritten as

$$\frac{1}{SF_j} \leq \frac{P_j r_{j,i(j)}^{-u}}{\left(m_{i(j)} + fI_o + \sum_{k \in H}\left(P_k r_{k,i(j)}^{-\mu}\right)\right) b_j} \leq \frac{P_j r_{j,i(j)}^{-u}}{\left(m_{i(j)} + fI_o + c_{i(j)}\right) b_j} \tag{6.24}$$

Now we define $a_{j,i(j)} = \frac{r_{j,i(j)}^{-u}}{\left(m_{i(j)}+fI_o+c_{i(j)}\right)b_j}$ in (6.24) and replace the revised (6.24) into the objective function z in (6.19), the dependence of the objective function to the decision variable SF_j is eliminated, and the problem is reduced to

$$\begin{cases} z = & \sum_{j \in H} \frac{B_c}{SF_j} \leq \sum_{j \in H} B_c a_{j,i(j)} P_j \\ \sum_{k \in H}\left(P_k r_{k,i}^{-\mu}\right) \leq & c_i \qquad i \in A \\ P_j \geq & 0 \qquad j \in H \end{cases} \tag{6.25}$$

(6.25) is a perfect linear programming problem, and can be solved by the popular simplex method. After the optimal power vector $\{P_j\}$ is obtained, we can drive the corresponding $\{SF_j\}$ from (6.20) by equalizing the inequality. With (6.25), it is easily seen that the handoff request should be denied if searching for an feasible solution failed, which means that the new mobile can not be supported without violating the BER requirements of other mobiles.

Unfortunately, the optimal solution for (6.25) only provides an upper bound for the problem described in (6.14) to (6.16), and sometimes can not be reached. The optimal power vector for the linear programming problem in (6.25) could contain zeros for some P_j; this is very common in the case that the number of constraints is less than the number of decision variables. However according to the constraints in (6.20), $SF_j P_j$ must be greater than a positive value, which is impossible if $P_j = 0$. Thus some additional steps must be taken in such cases. In the next section, we will discuss the heuristic algorithm that seeks the sub-optimal solution of (SF_j, P_j) for all the handoff mobiles.

6.4.3 Proposed Sub-Optimal SF/POWER Decision Algorithm

From the previous discussion, we know that the optimal solution for linear programming problem (6.25) might not results in a feasible solution for the original problem due to the possible zero-value power assignment. Furthermore, the allowed Spreading Factor in reality can only be taken from a limited set of positive integer. This necessitates a post-processing to convert the optimal vector to a feasible solution. For example, if the spreading factor has a value of 35.5, we should replace it with the closest greater integer value that is in the feasible solution space. Besides this, due to the background noise, we must also make sure that the signal-to-noise-ratio P_j/n be greater than a threshold, say $10 \sim 15$ db. These considerations lead us to a hybrid algorithm which utilizes the linear programming methods and additional constraints re-evaluation from the original problem. The algorithm is named HYB.

Given the input of the original problem in (6.14 to 6.16) and the reduced linear programming problem in (6.25), the algorithm must go through a series of steps. First, we will find the optimal power vector for problem (6.25), using a standard simplex method. This is followed by a check to see if there are any P_j in the solution vector equal to zero or the respective SNR is below the threshold.

If the above condition is not satisfied, we can move ahead to obtain a corresponding SF_j using (6.20). The resultant SF_j is usually a non-integer positive value, which will be further processed to result in a larger SF_j in the feasible space. The new vector of SF_j is substituted in to (6.20) again, and results in a new P_j vector. The new P_j vector is less than the first power vector we obtained for (6.25), since we are actually using a larger SF_i. At this stage, we have all the decision variables needed and the algorithm is terminated.

Otherwise, a transform should be done which will further simplify the linear problem in (6.25). We then fix the problematic decision variables to the minimal

power level and replace them back into (6.25). This will generate a new linear programming problem with reduced number of decision variables. The revised problem will be processed again using the simplex method, and the same check process will be applied to the the new optimal vector. It can be see that the algorithm will eventually terminate, and the maximum loop time will be less than the number of handoff mobiles. The pseudo code for the above procedure is described below:

Procedure Non_Linear_Simplex
Input :
b_j, **the desired SIR for** j^{th} **handoff mobile**
$r_{j,i}$ **the estimated distance between** j^{th} **mobile and** i^{th} **cell**
m_i, **the load of cell i;**
SF_t, **the least SF for traffic type t;**
b_t, **the least SIR for traffic type t;**
Known system parameters:
u : *path loss degree*;
SF_{max} : *the maximum possible spreading factor*;
f : $other - cell - interference\ ratio$;
ϕ : *the minimum possible power level*;

/*calculate some convenient parameters:*/
$I_o = \frac{f}{\mid A \mid} \sum_{u \in A} m_u$
FOR $i \in A$ **and** $j \in H$ **DO**
$a_{j,i} = \frac{r_{j,i}^{-u}}{(m_i + fI_o + c_i)b_j}$;
$c_i = \frac{SF_t}{b_t} - m_i - fI_o$;
END FOR

The decision vector is $\bar{P} = \{P_j\}$
and $\bar{SF_j} = \{SF_j\}$;
iteration:
use simplex method on

$$z = \sum_{j \in H} B_c\ a_{j,i}\ P_j \qquad (*)$$
$$\sum_{k \in H} \left(P_k\ r_{k,i}^{-\mu}\right) \leq c_i$$

(**)

$$P_j \geq \&\ 0 \qquad j \in H$$

and let $\hat{P} = \{\hat{P}_1, \hat{P}_2, ...\hat{p_J}\}$ **be the result.**

calculate $\mathcal{P} = \{j |\ \hat{P}_j < \phi,\ \hat{P}_j \in \hat{P}\ \}$
IF $\mathcal{P}$ == *empty set* **THEN**
case1:

```
        calculate SF_j using the equation:
            ŜF_j = 1/(P̂_j a_{j,i})    for j ∈ P;
          GOTO end;
    ELSE
case2:
        for each j ∈ P
                P̂_j = max{φ, 1/(a_{j,i} SF_max)}
        end for
        replace the value of vector P̄_P into (*) and (**)
        reduce the decision vector:
                P̄ = P̄ − P̄_P;
                S̄F = {SF_j | j ∈ P̄};
          GOTO iteration;
    END IF
end:
 /*finalize the spreading factor and power level vector*/
     FOR j ∈ H DO
         SF*_j = max{round(ŜF_j), 16⌈ŜF_j/16⌉};
         calculate :
                P*_j = 1/(SF*_j a_{j,i});
         calculate the objective function as (*);
End Procedure
```

6.4.4 Numerical Results and Performance Discussion

To evaluate the performance of HYB algorithm, we examined handoff scenarios with different traffic. Our numerical experiments assume the two-cell handoff situation, where the handoff traffic is from the original cell (C_o) to the target cell (C_t). C_o and C_t have 5 and 10 in-cell mobiles kept active all the time, respectively. We assume at least one mobile for each traffic type exist in each cell, thus the sum of the transmission power of handoff traffic is most strictly limited. The handoff traffic contain 10 mobiles, distributed in the 450-meter handover area with equal distance. The performance results to be examined have 4 parts: (1) overall throughput for handoff mobile, (2) the assignment of SF, (3) the power assignments, and (4) the BER in each handoff mobile. To investigate the relevance of the performance with the location of handoff mobiles, we studied the results for different location offset

of the handoff mobiles. A location configuration of **offset10** means that the first handoff mobile is 10 meters from the leftmost point of handoff area, and the next mobile will be 50 meter to the right, so on and so forth. In our experiments, up to fifty different offset configurations are studied.

In the first case of our experiments, all handoff mobiles carry WWW traffic. The numerical results are shown in Figure 6.7a and 6.7c. In the SF assignment for the **offset0**, all handoff mobiles are assigned SF=256 except for the 8^{th} mobile, which is assigned SF=32. For the handoff mobile with higher SF value, the corresponding power level is low (in the range of 1 to 3). The 8^{th} mobile has a power level of 11, which compensates the potential high BER when using such a small SF. The other location configurations show similar results, except that the first mobile is assigned a spreading factor between 32 and 256. The BER of the handoff mobile is satisfied in all mobile locations within the handoff area. Most of the BER concentrate on the range between 0.00001 to 0.000045, where the worst BER is observed at the 8^{th} mobile. The close-to-desired BER performance shows that the system resources are shared equally. The overall throughput in the handoff area is maintained from 380 to 400 kbps. This is about 20% increase compared to the case without HYB algorithm, where the SF of each handoff mobile is fixed to 160, resulting in a throughput of 312 kbps.

The second experiment was performed for all video handoff traffic. We observed that the first and the 8^{th} handoff mobiles are assigned small spreading factors. Other mobiles still use the largest SF (=256). This corresponds to the two peak power level in the power assignment. For instance, the **offset0** configuration have a SF assignment of *(32, 256, 256,256, 256, 256, 256,32,256,256)*, and the power vector of *(6.4, 1, 1.3, 1.4 ,1.5, 1.4, 1.5, 15.2, 2.4, 2)*.

The BER level of the handoff mobiles are also controlled below the maximum allowed value. The highest BER is also reached at the 8^{th} mobile, with BER=0.00031.

The lowest BER is 0.0001. The throughput for video traffic can vary from 600 kbps to 950 kbps, depending on the location of handoff mobiles. Without the HYB algorithm, the handoff algorithm has to use a more conservative strategy to choose the SF by considering them as in-cell traffic in the target cell. This will result in a SF of 112 for all handoff mobiles, corresponding to a throughput of 446.4 kbps. Thus, even the worst case of HYB can benifit from a 26% performance gain.

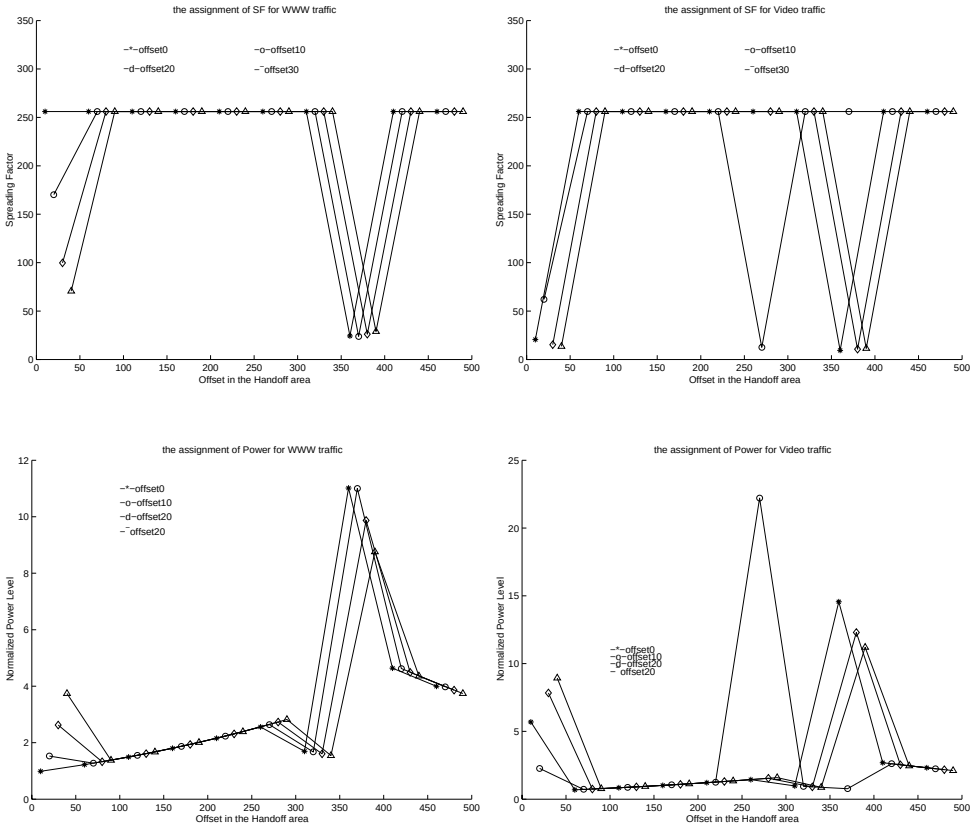

Figure 6.7: Performance for video traffic: (a)SF assignment for WWW traffic, (b)SF assignment for video traffic, (c)power level for WWW traffic, and (d) power level for video traffic

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

The success of distributed multimedia system relies on the solutions to the three key aspects: (1) video encoding algorithms, (2) video decoding algorithms, and (3) network transportation protocols that guarantee the delivery of video stream timely and error-free. In the first part of this dissertation, we investigated a generic, pure-software-based, parallel decoding algorithms. Built on a master/slave architecture, we parallelized the sequential MPEG-2 decoding algorithm in data-partition and data-pipeline manner.

For the data-partition scheme, the decoding task is parallelized in the frame level. In order to reduce the communication overhead, we proposed a quick partition algorithm, which can find the close-to-optimal partition efficiently. The data-partition scheme is shown to perform well in the small and moderate settings, but not as scalable as the data pipeline scheme, due to the overhead in master node and high-bandwidth requirement. The data pipeline scheme is able to produce high gain from parallel processing with little overhead. This is made possible by the frame-level parallelism and the reduction of referencing data through optimizing the block size. Meanwhile the overhead in the master node is kept with a low degree by adopting a quick bitstream analysis.

In the second part of this dissertation, we proposed a MAC layer protocol for WCDMA system that supports multimedia traffic with a guaranteed quality. The proposed protocol also maximizes the number of users supported. The major contributions are:

- The admission control is investigated in detail.

- The results show that even though the number of users increases, the average BER is maintained below the upper limits.
- Improvement schemes are proposed to reduce the admission delays.
- A new scheduling algorithm is discussed to improve the system throughput
- The dynamic spreading factor protocol is extended to multi-cell environment and a new soft handoff algorithm is proposed.
- A batching method is studied to reduce the handoff delay, particularly the update time.
- A join-optimization algorithm is proposed for the allocation of spreading factor and transmitting power for the handoff mobile stations.

Our studies provided solutions and recommendation for some important aspects in distributed multimedia systems. However, there are still much work ahead to fulfill the ultimate goal: instant access to high quality multimedia information from anywhere, at anytime. In my future research, I will continue my efforts approaching this goal. Among many potential topics in the distributed multimedia system, I believe that the following three issues need to be addressed: (1) how to provide video-on-demand in WCDMA downlink and support large number of users? (2) how to real-time decode MPEG-4 document which consists of several media streams, and (3) how to make the video encoding process more accurate in bit allocation and adaptive to bitrate fluctuation. Some further elaboration for the above topics are shown in the following sections.

7.1 Improve Transportation Layer Performance in WCDMA downlink

The HSDPA (High Speed Downlink Packet Access) of 3GPP (third generation pattern-ship project) initiates the efforts to provide high speed downlink for multimedia traffic, in particular video traffic. With dynamic spreading factor, adaptive modulation (AMC) and link level re-transmission, the achievable bandwidth could be upto 10 Mbps. With HSDPA, many video application is becoming more and more realistic, such as news-on-demand and video-on-demand. However, studies also show

that the system performance will degrade significantly when mobile units move away from base station, and mobiles in the boundary area of the cell suffer the most.

From the transportation point of view, the link-level re-transmission will directly increase the delay of upper layer ACK, which could trigger the rate control algorithm built-in the transportation layer. For example, if the delay of the upper layer ACK is larger than the TCP time-out value, the source transmission window will be reduced by half (as in TCP Tahoe). Due to the congestion avoidance consideration in the source, the transmitting rate will not be recovered immediately even the link condition is improved shortly.

We believe that transportation layer should be isolated from the link-level fluctuation. This is particularly important to the delay sensitive applications, such as streaming video, audio on demand, and interactive multimedia applications. More effective methods can be investigated to provide a smooth transportation layer upon the dynamic of MAC layer service. Particularly, the feasibility of transportation layer buffering schemes in the base station or mobile switching center can be investigated in the near future.

7.2 Parallel MPEG-4 Decoding

The parallel decompression of MPEG-4 stream will introduce another set of problem. Unlike in the case of MPEG-2, the MPEG-4 video consists of several sub-streams that require different decoding procedure , as well as the decoding complexity. It could contain graphic object, natural video, animation, texture background and other media stream. These sub-streams is synchronized both inside and outside with each other. Thus a parallel MPEG-4 decoder need to schedule the decoding subtask such that the system load is balanced. Moreover, MPEG-4 introduce much more interactivity between the end-user and content provider, which make it important to have a quick response time. As we can image, there is a lot of ways to parallelize the

decoding of MPEG-4, how it can be approached effectively will be explored in our future work.

7.3 Optimal Rate Distortion Control in Video Encoding

Ideally, QoS-guaranteed communication (e.g., ATM [6] and IPv6 [7]) should be used for multimedia transportation. However, we speculate that majority of streaming video will still be provided on the best-effort network (e.g.,INTERNET) in the near future. The end-to-end video quality can be greatly affected due to the dynamic of available bandwidth. Thus it is essential that the video encoding makes the best use of the available bandwidth, and generate the compressed bitstream with highest quality as possible. Bandwidth adaptive video encoding technology, dynamically update the video encoding parameters according to the current network situation, is expected to deliver better video quality with limited bandwidth. Some related work on this area can be found in [49–52]. These works have their focus on adjusting the quantization parameter of the DCT coefficients based on some particular rate-distortion model. However, their model are based on the fixed image resolution, which impose an unnecessary limitation to further exploit the rate-distortion theory and result in poor video quality when bit budget is low. When the available data rate is too low, a large quantization scale will be used in the traditional rate control methods, in order to maintain the desired bit rate. The larger quantization scale degrade the image quality severely, since detail image structure can not survive a coarse quantization.

From our preliminary experimental results, we observed that video quality can be significantly improved by re-rendering the image into a small resolution and use a relative precise quantization scale. The tradeoff here is: when the bit budget runs low, we can either down-sample image to a smaller resolution and use a rather precise quantization parameter, or directly adapt to a coarse quantization scale with the original image size. Therefore a big challenge is: for a given bit rate,

how should we decide the best image resizing factor and the quantization parameters to reconstruct the optimal video quality. The answer to this question requires a new rate-distortion model that can model the video contents and resizing factor. We are currently investigating the accuracy of several candidate models, and experiments are under development to verify our models.

APPENDIX A
C SOURCE CODE FOR PARALLEL MPEG-2 DECODER

```
/*****************************************
data partition algorithm
*****************************************/
extern int totalpicturenum;
extern int workstationnum;
extern int First_Picture;
extern int myrank;
void DistributeData_Partition()
{ int i,j;
  int code;
  totalpicturenum++;
  frameptr[0]->lpFile=frameptr[0]->rawData;
      while(1){
            Global_next_start_code();
            code = Global_Show_Bits32();
            if (code==PICTURE_START_CODE)break;
      }
      j=0;
      do{
            frameptr[0]->lpFile[j]=Global_Get_Byte();
            j++;
      }while(Global_Show_Bits32()!=PICTURE_START_CODE);
```

```
      frameptr[0]->ld->Incnt=32;
      frameptr[0]->ld->Rdptr = frameptr[0]->ld->Rdbfr + 2048;
      frameptr[0]->ld->Rdmax = frameptr[0]->ld->Rdptr;
      frameptr[0]->ld->Bfr = 0;
      frameptr[0]->rawsize=j;
      frameptr[0]->picturenum=totalpicturenum;
      frameptr[0]->Float_Counter=0;
      totalpicturenum++;
      printf("the size of raw sending picture is %d  the actual
         size of this packets is %d\n",sizeof(struct PictureBuffer),j);
      for(i=0;i<workstationnum;i++)
            MPI_Send (frameptr[0],sizeof(struct PictureBuffer)
/workstationnum, MPI_BYTE, i+1, 100, MPI_COMM_WORLD);
      }

int video_sequence_Partition(Bitstream_Framenumber)
int *Bitstream_Framenumber;
{
  int Return_Value,i;
  int sizeL,sizeC;
  int lastframe=0;
  int status;
  int rett;
  clock_t wj_tmp1,wj_tmp2;
  struct tms tbuf;
  int RCVsize;
  Bitstream_Framenum = *Bitstream_Framenumber;
```

```
  Sequence_Framenum=0;
  RCVsize=(frameptr[0]->horizontal_size*frameptr[0]->vertical_size)
   /workstationnum;
  Acc_Time=0;
  First_Picture=1;
  for(;;){
int j=0;
        wj_tmp1=times(&tbuf);
DistributeData_Partition();
if(Sequence_Framenum==0)decodeframe(frameptr[0],0);
for(j=0;j<workstationnum;j++)
{
           rett=MPI_Recv (frame_pool[0][0]+j*RCVsize, RCVsize,
              MPI_CHAR,j+1,100, MPI_COMM_WORLD, &status);
}
Sequence_Framenum++;
        wj_tmp2=times(&tbuf);
        Acc_Time+=(wj_tmp2-wj_tmp1);
printf("sequence Framenum=%d",Sequence_Framenum);
        printf("Master Acc Time is %f secs, FPR=%f\n",(float)Acc_Time
  /CLK_TCK,(float)Sequence_Framenum*CLK_TCK/Acc_Time);
}}


void receiveDistributeData_Partition()
{
        int i,j,actread;
        int code;
```

```
        int status;
        char * tmp;
        i=0;
        tmp=frameptr[0];
        MPI_Recv (tmp, sizeof(struct PictureBuffer)/workstationnum,
MPI_BYTE, 0, 100, MPI_COMM_WORLD, &status);
        if (((struct PictureBuffer *)frameptr[0])->mb_width == -1)
        {
             frameptr[0]->mb_width=-1;
             return;
        }
//receive raw data from the master
        frameptr[0]->lpFile=frameptr[0]->rawData;
        frameptr[0]->ld=&(frameptr[0]->base);
        frameptr[0]->ld->Incnt=32;
        frameptr[0]->ld->Rdptr = frameptr[0]->ld->Rdbfr + 2048;
        frameptr[0]->ld->Rdmax = frameptr[0]->ld->Rdptr;
        frameptr[0]->ld->Bfr = 0;
        frameptr[0]->Float_Counter=0;
}
void sendPictureData_Partition()
{
        int i,j;
        int code;
        int sizeL;
        int sizeC;
        i=0;
```

```
        sizeL=(frameptr[0]->Coded_Picture_Width*frameptr[0]->
Coded_Picture_Height)/workstationnum;
        MPI_Send (Dithered_Image[0],sizeL , MPI_BYTE, 0, 100,
               MPI_COMM_WORLD);
}
void  decodeframe_Partition(struct PictureBuffer * frame,int mrank)
{        int Return_Value;

/*assume the frame rate for one node is 5f/s, it take
  200ms for one picture, so sleep 200/workstationnum ms*/
       Return_Value=Headers(frame);// ret =1 mean picture_head meeted
       if (Return_Value==0)return -1;
       else if (Return_Value!=1) return -1;
       if (First_Picture){
            Initialize_Sequence(frame);
            First_Picture=0;
            if(myrank==0)}
return;
}
/*********************************************
master node main(), data pipeline scheme
*********************************************/
void main(argc,argv)
int argc;
char *argv[];
{
  int ret, code;
```

```
int count;
int i,j;
int index;
int sos = 1;
int dummyargc;
int size;
FILE * tmpfp,*grpfp;
grpfp=fopen("/amd/birch/export/class00/cen4500fa00/yhe/
   mpeg2dec12/tm.pg","r");
printf("grpfile:%d\n",grpfp);
fscanf(grpfp,"#%d %d\n",&FAKESCALE_BMP,&FAKESCALE_NET);
fclose(grpfp);
decodestate=START;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
MPI_Comm_size( MPI_COMM_WORLD, &workstationnum );
workstationnum--;
 //the master will should be exculded from the slave num;
sprintf(logfilename,"e:\\temp\\logfile%d.txt",myrank);
logfp=fopen(logfilename,"w");
if (myrank == 0)               /* I am the master */
{
    printf("MAster processing up!\n");
    D_Float_Counter = 0;
    D_Info_Flag     = 1;
    for (count =0; count < argc; count++) {
printf("the argc is %d ;; the argv[%d] is %s\n",
```

```
  count, count, argv[count]);
    }
    Clear_Options();
/* decode command line arguments */
    Process_Options(dummyargc,dummyargv);
    for(i=0;i<PARALLELSIZE;i++)
    {
frameptr[i]=malloc(sizeof(struct PictureBuffer));
if (!frameptr[i])Error("frameptr malloc error!\n");
    }
    for(i=0;i<PARALLELSIZE;i++)
      frameptr[i]->ld = &(frameptr[i]->base);
     /* select base layer context */
     /* open MPEG base layer bitstream file(s) */
    lpFile=(unsigned char *)malloc(6553600);
    tmpfp=fopen("mei60f.m2v","r");
    fread(lpFile,1,SUPER_BUF_SIZE,tmpfp);
    fclose(tmpfp);
    Float_Counter = 0; /* SUPER GLOBAL  */
    Last_Round    = 0;
    Last_Frame = 0;
    D_Info_Flag   = 1; /* We set this to let the
         displayer have the infor at the first time */
    InitiateTime();
    Global_Bfr=0;
    Global_Incnt=0;
    Float_Counter=0;
```

```
      do
      {
        Global_Bfr |=((unsigned char) lpFile[Float_Counter]) <<
            (24 - Global_Incnt);
Float_Counter++;
        Global_Incnt += 8;
      }
      while (Global_Incnt <= 24);
      if(Global_Show_Byte()==0x47)
      {
         sprintf(Error_Text,"Decoder currently does not
           parse transport streams\n");
         Error(Error_Text);
       }
      Global_next_start_code();
      code = Global_Show_Bits32();
      switch(code)
      {
        case SEQUENCE_HEADER_CODE:
             break;
        case PACK_START_CODE:
  /* System_Stream_Flag = 1; */
  printf("we don't support system stream \n");
        case VIDEO_ELEMENTARY_STREAM:
      /* System_Stream_Flag = 1; */
  printf("we don't support system stream \n");
         break;
```

```
        default:
          sprintf(Error_Text,"Unable to recognize stream type\n");
          Error(Error_Text);
          break;
        }
      Initialize_Decoder();
      ret = Decode_Bitstream();
  }/*main process end*/
  else{
       int mydecodednum;
       struct tms tbuf;
       clock_t wj_tmp1,wj_tmp2;
       printf("this is slaver with rank %d\n",myrank);
       fflush(NULL);
       Clear_Options();
  /* decode command line arguments */
       Process_Options(dummyargc,dummyargv);
       for(i=0;i<PARALLELSIZE;i++)
       {
  frameptr[i]=malloc(sizeof(struct PictureBuffer));
  if (!frameptr[i])Error("frameptr malloc error!\n");
        }
       for(i=0;i<PARALLELSIZE;i++)frameptr[i]->ld =
&(frameptr[i]->base); /* select base layer context */
       Float_Counter = 0; /* SUPER GLOBAL  */
       Last_Round    = 0;
       Last_Frame = 0;
```

```
        D_Info_Flag   = 1; /* We set this to let
          the displayer have the infor at the first time */
        first_dither=1;
        InitiateTime();
        Initialize_Decoder();
        basenum=0;
        rfp=fopen(rfpfilename,"wb");
        pfp=fopen(pfpfilename,"rb");
        First_Picture=1;
        picorder=0;
        mydecodednum=0;
        receiveDistributeData_Partition();
        for(j=0;j<60;j++){
  picorder=j*PARALLELSIZE*workstationnum+(myrank-1)*PARALLELSIZE;
  mydecodednum=j*PARALLELSIZE;
  wj_tmp1=times(&tbuf);
  printf("do the %d to %d\n",picorder,picorder+PARALLELSIZE);
  //Extract Data
          decodeframe_Partition(frameptr[0],myrank);
  wj_tmp2=times(&tbuf);
  sExtractAcc_Time+=(wj_tmp2-wj_tmp1);
  printf("The extract acc time is %d\n", sExtractAcc_Time);
//Send result back
  if (first_dither)
  {
        first_dither=0;
        dither_initial();
```

```
  }
  //Dither the group of picture. the result is window bmp pic stored in
  //global Dithered_Image
  if (frameptr[0]->progressive_sequence){
    Dither_Frame_Partition(myrank);
  }else{
    if ((frameptr[0]->picture_structure == FRAME_PICTURE &&
      frameptr[0]->top_field_first) ||
      frameptr[0]->picture_structure == BOTTOM_FIELD)
    {
Dither_Top(i,Dithered_Image[i]);
Dither_Bottom(i,Dithered_Image2[i]);
}else{
  Dither_Bottom(i,Dithered_Image[i]);
     Dither_Top(i,Dithered_Image2[i]);
}
    }
  wj_tmp2=times(&tbuf);
  sDithAcc_Time+=(wj_tmp2-wj_tmp1);
  printf("The Dither acc time is %d\n", sDithAcc_Time);
  wj_tmp1=times(&tbuf);
  sendPictureData_Partition();
  wj_tmp2=times(&tbuf);
  sSendAcc_Time+=(wj_tmp2-wj_tmp1);
//Recieve Raw Data
  if (frameptr[0]->mb_width==-1)break;
  receiveDistributeData_Partition();
```

```
//RECEIVE raw data from master, after get the data, it
   fflush(NULL);
  }
  fclose(pfp);
  fclose(rfp);
  fclose(logfp);
}
  fclose(logfp);
  MPI_Finalize( );
  //ExitProcess(0);
  return ;
}

void Initialize_Sequence(frame)
struct PictureBuffer * frame;
{
int cc, size;
int i,k;
  int MBAmax;
  static int Table_6_20[3] = {6,8,12};
  /* force MPEG-1 parameters for proper decoder behavior */
  /* see ISO/IEC 13818-2 section D.9.14 */
for(i=0;i<PARALLELSIZE;i++)
{
  if (!frame->base.MPEG2_Flag)
  {
    frameptr[i]->progressive_sequence = 1;
```

```
    frameptr[i]->progressive_frame = 1;
    frameptr[i]->picture_structure = FRAME_PICTURE;
    frameptr[i]->frame_pred_frame_dct = 1;
    frameptr[i]->chroma_format = CHROMA420;
    frameptr[i]->matrix_coefficients = 5;
  }
  /* round to nearest multiple of coded macroblocks */
  frameptr[i]->mb_width = (frameptr[i]->horizontal_size+15)/16;
  frameptr[i]->mb_height =(frameptr[i]->base.MPEG2_Flag &&
!frameptr[i]->progressive_sequence) ?
2*((frameptr[i]->vertical_size+31)/32)
        : (frameptr[i]->vertical_size+15)/16;
  frameptr[i]->Coded_Picture_Width = 16*frameptr[i]->mb_width;
  frameptr[i]->Coded_Picture_Height = 16*frameptr[i]->mb_height;
  /* ISO/IEC 13818-2 sections 6.1.1.8, 6.1.1.9, and 6.1.1.10 */
  frameptr[i]->Chroma_Width = (frameptr[i]->
        chroma_format==CHROMA444) ?
        frameptr[i]->Coded_Picture_Width
        : frameptr[i]->Coded_Picture_Width>>1;
  frameptr[i]->Chroma_Height = (frameptr[i]->
        chroma_format!=CHROMA420) ?
        frameptr[i]->Coded_Picture_Height
        : frameptr[i]->Coded_Picture_Height>>1;
  frameptr[i]->block_count = Table_6_20[frameptr[i]->
     chroma_format-1];
  for (cc=0; cc<3; cc++)
  {
```

```
    if (cc==0)
      size = frameptr[i]->Coded_Picture_Width*
frameptr[i]->Coded_Picture_Height;
    else
      size = frameptr[i]->Chroma_Width*frameptr[i]->Chroma_Height;
    if (!(frame_pool[i][cc] = (unsigned char *)malloc(size)))
      Error("frame_pool[K] malloc failed\n");
  }
  MBAmax=frameptr[i]->mb_width*frameptr[i]->mb_height;
  if(!(frameptr[i]->data=malloc(MBAmax*sizeof(struct MBlock))))
  Error("framptr->data malloc error\n");
  global_microblocks[i]=frameptr[i]->data;
}
  for (cc=0; cc<3; cc++)
  {
forward_reference_frame[cc]=frame_pool[0][cc];
backward_reference_frame[cc]=frame_pool[0][cc];
    if (frame->base.scalable_mode==SC_SPAT)
    {
      /* this assumes lower layer is 4:2:0 */
      if (!(llframe0[cc] = (unsigned char *)
malloc((lower_layer_prediction_horizontal_size*
lower_layer_prediction_vertical_size)/(cc?4:1))))
        Error("llframe0 malloc failed\n");
      if (!(llframe1[cc] = (unsigned char *)
malloc((lower_layer_prediction_horizontal_size*
lower_layer_prediction_vertical_size)/(cc?4:1))))
```

```
        Error("llframe1 malloc failed\n");
    }
  }
  /* SCALABILITY: Spatial */
  if (frame->base.scalable_mode==SC_SPAT)
  {
    if (!(lltmp = (short *)malloc
(lower_layer_prediction_horizontal_size*
((lower_layer_prediction_vertical_size*
vertical_subsampling_factor_n)/
vertical_subsampling_factor_m)*sizeof(short))))
      Error("lltmp malloc failed\n");
  }
}

//master func : data pipeline version

void DistributeData()
{
int i,j;
int code;
for (i=0;i<PARALLELSIZE;i++)
{
//totalpicturenum++;
frameptr[i]->lpFile=frameptr[i]->rawData;
while(1){
Global_next_start_code();
```

```
code = Global_Show_Bits32();
if (code==PICTURE_START_CODE)break;
}
j=0;
do{
frameptr[i]->lpFile[j]=Global_Get_Byte();
j++;
}while(Global_Show_Bits32()!=PICTURE_START_CODE);
frameptr[i]->ld->Incnt=32;
frameptr[i]->ld->Rdptr = frameptr[i]->ld->Rdbfr + 2048;
frameptr[i]->ld->Rdmax = frameptr[i]->ld->Rdptr;
frameptr[i]->ld->Bfr = 0;
frameptr[i]->rawsize=j;
frameptr[i]->picturenum=totalpicturenum;
frameptr[i]->Float_Counter=0;
totalpicturenum++;
MPI_Send (frameptr[i],sizeof(struct PictureBuffer),
MPI_BYTE, ((basenum/PARALLELSIZE)%workstationnum)+1,
100, MPI_COMM_WORLD);
    }
}
void TerminateSlave()
{
int i;
printf("Terminate Slave!\n");
//getch();
frameptr[0]->mb_width=-1;
```

```
for(i=1;i<=workstationnum;i++){
printf("Terminate slave %d\n",i);
MPI_Send (frameptr[0],sizeof(struct PictureBuffer),
MPI_BYTE, i, 100, MPI_COMM_WORLD);
}
}

//slave func
void receiveDistributeData()
{
    int i,j,actread;
    int code;
    int status;
    char * tmp;
    i=0;
    while(1)//for (i=0;i<PARALLELSIZE;i++)
    {
if (i>=PARALLELSIZE)break;
jjj=i;
tmp=frameptr[i];
MPI_Recv (tmp, sizeof(struct PictureBuffer),
MPI_BYTE, 0, 100, MPI_COMM_WORLD, &status);
i=jjj;
if (((struct PictureBuffer *)frameptr[i])->mb_width == -1)
{
frameptr[0]->mb_width=-1;
printf("jjj = %d\n",jjj);
```

```
return;
}
//receive raw data from the master
frameptr[i]->lpFile=frameptr[i]->rawData;
frameptr[i]->ld=&(frameptr[i]->base);
frameptr[i]->ld->Incnt=32;
frameptr[i]->ld->Rdptr = frameptr[i]->ld->Rdbfr + 2048;
frameptr[i]->ld->Rdmax = frameptr[i]->ld->Rdptr;
frameptr[i]->ld->Bfr = 0;
frameptr[i]->Float_Counter=0;
i++;
}
}
char picparam[100];
//slave func
//make the bmp picture to send, instead of y,u,v data
void sendPictureData()
{
int i,j;
int code;
int sizeL;
int sizeC;
i=0;
while(1)//for (i=0;i<PARALLELSIZE;i++)
{
if (i>=PARALLELSIZE)break;
sizeL=(frameptr[i]->Coded_Picture_Width*
```

```
          frameptr[i]->Coded_Picture_Height)/FAKESCALE_NET;
sizeC = (frameptr[i]->Chroma_Width*
           frameptr[i]->Chroma_Height)/FAKESCALE_NET;
sprintf(picparam, "%d %d %d %d %d %d %d %d %d %d",
    frameptr[i]->matrix_coefficients, frameptr[i]->
    Coded_Picture_Width, frameptr[i]->Coded_Picture_Height,
            frameptr[i]->Chroma_Width,
    frameptr[i]->Chroma_Height, frameptr[i]->chroma_format,
    frameptr[i]->progressive_sequence,
            frameptr[i]->picture_structure,
    frameptr[i]->top_field_first,frameptr[i]->picture_coding_type);
//FIRSTsend the picture param, then send picture bmp data
printf("send picture %d\n",i);
if (frameptr[i]->progressive_sequence)
{
MPI_Send (picparam,100, MPI_CHAR, 0, 100, MPI_COMM_WORLD);
MPI_Send (Dithered_Image[i],sizeL , MPI_BYTE,
0, 100, MPI_COMM_WORLD);
}
else
{
rett=MPI_Send (picparam,100, MPI_CHAR, 0, 100, MPI_COMM_WORLD);
            rett=MPI_Send (Dithered_Image[i],sizeL ,
MPI_BYTE, 0, 100, MPI_COMM_WORLD);
}
i++;
}
```

```
}
int decodeframe(struct PictureBuffer * frame,int basenum)
{
int Return_Value;
Return_Value=Headers(frame);// ret =1 mean picture_head meeted
if (Return_Value==0)return -1;
else if (Return_Value!=1) return -1;
if (First_Picture){
Initialize_Sequence(frame);
First_Picture=0;
if(myrank==0)
    }
getframe(frame,basenum);
return 0;
}
void showdecodepicture(int basenum)
{
decodehighhalf(frameptr[(basenum)%PARALLELSIZE]);
decodelowhalf(frameptr[(basenum)%PARALLELSIZE]);
if(myrank==0)frame_reorder(frameptr[(basenum)%PARALLELSIZE],
            Bitstream_Framenum, Sequence_Framenum);
if (!frameptr[(basenum)%PARALLELSIZE]->Second_Field)
{
Bitstream_Framenum++;
Sequence_Framenum++;
}
}
```

```
static int video_sequence(Bitstream_Framenumber)
int *Bitstream_Framenumber;
{
  int Return_Value,i;
  int sizeL,sizeC;
  int lastframe=0;
  int status;
int rett;
clock_t wj_tmp1,wj_tmp2;
struct tms tbuf;
  Bitstream_Framenum = *Bitstream_Framenumber;
  Sequence_Framenum=0;
  pfp=fopen(pfpfilename,"wb");
    DistributeData();
  decodeframe(frameptr[basenum%PARALLELSIZE],basenum);
  basenum+=PARALLELSIZE;
  for (i=0;i<workstationnum-1;i++)
   //file the No 2 to No (N-1) slave
  {
  DistributeData();
  basenum+=PARALLELSIZE;
  }
  for(;;)
  {
  wj_tmp1=times(&tbuf);
  i=0;
  while(1)//for(i=0;i<PARALLELSIZE;i++)
```

```
  {
  if (i>=PARALLELSIZE)break;
  jjj=i;
  printf("receive picture %d\n",i+basenum);
//Receive a picture
  rett=MPI_Recv (picparam, 100, MPI_CHAR,
(((basenum-(workstationnum)*PARALLELSIZE)/
PARALLELSIZE)%workstationnum)+1, 100,
MPI_COMM_WORLD, &status);
sscanf(picparam,"%d %d %d %d %d %d %d %d %d %d",
  &(frameptr[0]->matrix_coefficients), &(frameptr[0]->
          Coded_Picture_Width),
          &(frameptr[0]->Coded_Picture_Height),
          &(frameptr[0]->Chroma_Width),
  &(frameptr[0]->Chroma_Height),
          &(frameptr[0]->chroma_format),
          &(frameptr[0]->progressive_sequence),
          &(frameptr[0]->picture_structure),
  &(frameptr[0]->top_field_first),
          &frameptr[0]->picture_coding_type);
sizeL=(frameptr[0]->Coded_Picture_Width*
          frameptr[0]->Coded_Picture_Height)/FAKESCALE_NET  ;
sizeC = (frameptr[0]->Chroma_Width*frameptr[0]->
          Chroma_Height)/FAKESCALE_NET;
if (frameptr[0]->progressive_sequence)
{
if(frameptr[0]->picture_coding_type==B_TYPE)
```

```
  MPI_Recv (frame_pool[0][0], sizeL, MPI_BYTE,(((basenum-
            (workstationnum)*PARALLELSIZE)/PARALLELSIZE)%
            workstationnum)+1 ,100, MPI_COMM_WORLD, &status);
else {
  i=jjj;
  MPI_Recv (frame_pool[1][0], sizeL, MPI_BYTE,(((basenum-
            (workstationnum)*PARALLELSIZE)/PARALLELSIZE)%
            workstationnum)+1 ,100, MPI_COMM_WORLD, &status);
}
    }else
    {
        rett=MPI_Recv (frame_pool[0][0], sizeL,
         MPI_BYTE,(((basenum-(workstationnum)*PARALLELSIZE
        )/PARALLELSIZE)%workstationnum)+1 ,100,
        MPI_COMM_WORLD, &status);
     }
    printf("Master Acc Waiting Time is %d\n",mRecieveAcc_Time);
//Display a picture
printf("Master Acc Display Time is %d\n",DisAcc_Time);
i++;
}
//Count the acc time
wj_tmp2=times(&tbuf);
Acc_Time+=(wj_tmp2-wj_tmp1);
printf("PARALLELSIZE=%d   interval=%f",PARALLELSIZE,
           ((float)(wj_tmp2-wj_tmp1))/CLK_TCK);
printf("Master Acc Time is %f secs, FPR=%f\n",
```

```
          (float)Acc_Time/CLK_TCK,
          (float)(PARALLELSIZE)*CLK_TCK/(wj_tmp2-wj_tmp1));
//Send raw data
DistributeData();
basenum+=PARALLELSIZE;
  }
  fclose(pfp);
  if (Sequence_Framenum!=0)
  {
printf("output the last frame \n");
    Output_Last_Frame_of_Sequence(frameptr[basenum],
      Bitstream_Framenum);
  }

  Deinitialize_Sequence(frameptr[0]);
  *Bitstream_Framenumber = Bitstream_Framenum;
  return(Return_Value);
errhead:
  exit(-1);
}

void gameover()
{
  TerminateSlave();
  fclose(pfp);
  Deinitialize_Sequence(frameptr[0]);
  fclose(logfp);
```

```
  printf("wait for finalize\n");
  MPI_Finalize( );
  //ExitProcess(0);
  exit(0);
}
```

APPENDIX B
MATLAB SOURCE CODE FOR WCDMA HANDOFF ALGORITHM

```
function [theoBER, theoBER_o, bs1ber, bs2ber]=MaD(nuser1,nuser2,
activeset,pw1,pw2,f,d)
%BER changing from 5-->10 users during handoff
%power controled by BS1
%nuser1=5;
%nuser2=10;
%return value:
% theoBER, the combined BER of handover mobile using
% main path power control
% theoBER_o, the combined BER for handover mobile using
% is-95 'or for down' power control
% bs1ber the ber observed in bs1 using main path power
% control
SF=[32:16:256]; %15 SFs
for i=1: 15,
if nuser1<nuser2
           bs1ber(i)=ber(nuser1,nuser2,64,1,f);
           bs2ber(i)=ber(nuser2,nuser1,64,1,f);

   if activeset==1
theoBER(i)=ber(nuser1, nuser2,SF(i), 1, f);
theoBER_o(i)=theoBER(i);
```

```
   elseif activeset==3
if d>1000
ppww1= ((2000-d)/d)^4;
ppww2=1;
else
ppww1=1;
ppww2=(d/(2000-d))^4;
end
theoBER_o(i)=min(ber(nuser1,nuser2,SF(i),ppww1,f),
   ber(nuser2,nuser1,SF(i),ppww2,f));
theoBER(i)=min(ber(nuser1,nuser2,SF(i),1,f),
   ber(nuser2,nuser1,SF(i),pw2,f));
bs2ber(i)=ber(nuser2+pw2,nuser1,64,1,f);
   else %active set = 2
theoBER(i)=ber(nuser2,nuser1,SF(i),1,f);
theoBER_o(i)=theoBER(i);
   end
elseif nuser1>nuser2
           bs1ber(i)=ber(nuser1,nuser2,64,1,f);
           bs2ber(i)=ber(nuser2,nuser1,64,1,f);
   if activeset==1
theoBER(i)=ber(nuser1, nuser2,SF(i), 1, f);
               theoBER_o(i)=theoBER(i);
   elseif activeset==3
               if d>1000
                       ppww1= ((2000-d)/d)^4;
                       ppww2=1;
```

```
                else
                        ppww1=1;
                        ppww2=(d/(2000-d))^4;
end
          theoBER_o(i)=min(ber(nuser1,nuser2,SF(i),ppww1,f),
ber(nuser2,nuser1,SF(i),ppww2,f));
  theoBER(i)=min(ber(nuser1,nuser2,SF(i),pw1,f),
ber(nuser2,nuser1,SF(i),1,f));
     bs1ber(i)=ber(nuser1+pw1,nuser2,64,1,f);
  else
        theoBER(i)=ber(nuser2,nuser1,SF(i),1,f);
                theoBER_o(i)=theoBER(i);
  end
     end
end
function BER=ber(nuser1, nuser2,sf, pw, f);
% calculate the BER given:
% local cell load---nuser1
% other cell load---nuser2
%  interested mobile's SF---sf
% interested mobile's relative power level
% the relative ratio of othercell interference--f
% [BER]=ber(5,10,64,2,0.3)
SIR=pw*sf/(nuser1+f*nuser2);
tem1=(nuser2*f+nuser1-1)/(3*sf);
term2=(nuser1+f*nuser2)/(2*sf*pw)+nuser1/(3*sf);
sf*pw/(nuser1+f*nuser2)
```

```
BER=normcdf(-term2^(-1/2),0,1);
% the improved time for updating in target cell
T_u=zeros(100);
T_u([2 3 4 6 8 11 15 16 19 22])=[37 28 29 31 43 56 70 91 104 107]
T_u([23 27 32 34 36 39 40 44 45])=
[118 212 147 159 161 175 176 190 191]
T_u([49 51 53 57 61 63 69 75 81 87])=
[215 217 229 423 247 259 295 331 367 403]
T_i=60
T_e=20
lambda(1)=500
lambda(3)=250
lambda(2)=125
waiting=0;
for k=1:2
 i=1;
while i<100
  if waiting==0
   T_h(k,i)=T_u(i)+T_i+T_e;
   waiting=max(T_h(k,i)-lambda(k),0);
   i = i+1
  else
       T_h(k,i)=T_u(i)+T_i+T_e+waiting;
    cnt=1;
    tmp=T_h(k,i);
    while (tmp>cnt*lambda(k)),
tmp = max(T_u(i:i+cnt))+T_i+T_e*cnt;
```

```
cnt = cnt+1;
    end
  if cnt>1
        T_h(k,i)=tmp;
    for ss=1:cnt
T_h(k,i+ss) = T_h(k,i) + (cnt-ss)*lambda(k)
    end
  end
    i = i+cnt
%   waiting=max(T_h(k,i)-lambda(k),0);
  end
end
end
figure
hold
plot([1:99],T_h(1,1:99),'y-',[1:99],T_h(2,1:99),'g-')
plot([1:99],T_h(1,1:99),'yo',[1:99],T_h(2,1:99),'g*')
%,[1:100],T_h(3,:),'r+')
text(10,1000,'-o-lambda1=2')
text(10,900,'-*-lambda2=8')
xlabel('Handoff Mobile to the Target Cell','FontSize',20)
ylabel('the Handoff Time T_{handoff} (in ms)','FontSize',20)
title('System Response Time with Batching Processing','FontSize',20)
function invm=invers(A)
m=size(A);
m=m(1);
d=deter(A);
```

```
for i=1 : m
  for j=1:m
    invm(i,j)=(-1)^(i+j)*deter(aug(A,j,i));
    invm(i,j)=invm(i,j)/d;
  end
end
%   modle :  z=aj*x'
% A1*x>=m1+fIo x1~x6 follow cell1
% A2*x>=m2+fIo x7~x10 follow cell2
% r1mu*p'<=10
% r2mu*p'<=10
% set bj to desired SIRs for traffic mixture
m1=5
m2=10
powervector=zeros(400,10);
throughput=zeros(1,400);
r1mu=zeros(400,10);
r2mu=zeros(400,10);
d=1
%the offset pattern repeat every 50 meters
for d=1:1:50
r1=mod(([800:50:1250]+d-800),400)+800;
r2=2000-r1
r1mu(d,:)=10^10*r1.^(-3.5)
r2mu(d,:)=10^10*r2.^(-3.5)
bj=[5 5    5    5  5   5    5    5     5    5  ]+4
aj1=r1mu(d,:)./(m1+0.3*m2+10).*bj
```

```
aj2=r2mu(d,:)./(m2+0.3*m1+10).*bj
aj(1:6)=aj1(1:6)
aj(7:10)=aj2(7:10)
%minimum SF=256
%constrain for handoff mobile on cell1
A1(1,:)=[r1mu(d,1)*256/bj(1) -r1mu(d,2:10)]
A1(2,:)=[-r1mu(d,1) r1mu(d,2)*256/bj(2) -r1mu(d,3:10)]
A1(3,:)=[-r1mu(d,1:2) r1mu(d,3)*256/bj(3) -r1mu(d,4:10)]
A1(4,:)=[-r1mu(d,1:3) r1mu(d,4)*256/bj(4) -r1mu(d,5:10)]
A1(5,:)=[-r1mu(d,1:4) r1mu(d,5)*256/bj(5) -r1mu(d,6:10)]
A1(6,:)=[-r1mu(d,1:5) r1mu(d,6)*256/bj(6) -r1mu(d,7:10)]
%constrain for handoff mobile on cell2
A1(7,:)=[-r2mu(d,1:6) r2mu(d,7)*256/bj(7) -r2mu(d,8:10)]
A1(8,:)=[-r2mu(d,1:7) r2mu(d,8)*256/bj(8) -r2mu(d,9:10)]
A1(9,:)=[-r2mu(d,1:8) r2mu(d,9)*256/bj(9) -r2mu(d,10)]
A1(10,:)=[-r2mu(d,1:9) r2mu(d,10)*256/bj(10)]
%right hand of handoff mobile
b(1:6)=m1+0.3*m2
b(7:10)=m2+0.3*m1
%other cell constrain
A1(11,:)=r1mu(d,:)
A1(12,:)=r2mu(d,:)
b(11)=12
b(12)=12
type='max'
c=aj
rel='>>>>>>>>>><<'
```

```
pp=simplex2p(type,c,A1,rel,b,0)
sf(1,d)=bj(1)*(b(1)+ r1mu(d,2:10)*pp(2:10))/(r1mu(d,1)*pp(1))
sf(2,d)=bj(2)*(b(2)+ r1mu(d,1)*pp(1)+
r1mu(d, 3:10)*pp(3:10))/(r1mu(d,2)*pp(2))
sf(3,d)=bj(3)*(b(3)+ r1mu(d,1:2)*pp(1:2)+
r1mu(d,4:10)*pp(4:10))/(r1mu(d,3)*pp(3))
sf(4,d)=bj(4)*(b(4)+ r1mu(d,1:3)*pp(1:3)+
r1mu(d,5:10)*pp(5:10))/(r1mu(d,4)*pp(4))
sf(5,d)=bj(5)*(b(5)+ r1mu(d,1:4)*pp(1:4)+
r1mu(d,6:10)*pp(6:10))/(r1mu(d,5)*pp(5))
sf(6,d)=bj(6)*(b(6)+ r1mu(d,1:5)*pp(1:5)+
r1mu(d,7:10)*pp(7:10))/(r1mu(d,6)*pp(6))
sf(7,d)=bj(7)*(b(7)+ r2mu(d,1:6)*pp(1:6)+
r2mu(d,8:10)*pp(8:10))/(r2mu(d,7)*pp(7))
sf(8,d)=bj(8)*(b(8)+ r2mu(d,1:7)*pp(1:7)+
r2mu(d,9:10)*pp(9:10))/(r2mu(d,8)*pp(8))
sf(9,d)=bj(9)*(b(9)+ r2mu(d,1:8)*pp(1:8)+
r2mu(d,10)*pp(10))/(r2mu(d,9)*pp(9))
sf(10,d)=bj(10)*(b(10)+ r2mu(d,1:9)*pp(1:9) )/(r2mu(d,10)*pp(10))
powervector(d,:)=pp';
throughput(d)=5000*([1 1 1 1 1 1 1 1 1 1]*(1./sf(:,d)))
end
%wwwpower.eps
figure
hold
plot([0:50:450]+10,powervector(1,:),'r-',
[0:50:450]+10,powervector(1,:),'r*')
```

```
plot([0:50:450]+20,powervector(11,:),'y-',
[0:50:450]+20,powervector(11,:),'yo')
plot([0:50:450]+30,powervector(21,:),'g-',
[0:50:450]+30,powervector(21,:),'gd')
plot([0:50:450]+40,powervector(31,:),'b-',
[0:50:450]+40,powervector(31,:),'b^')
title('the assignment of Power for WWW traffic')
ylabel('Normalized Power Level')
xlabel('Offset in the Handoff area')
 text(100,11,'-*-offset0')
 text(100,10.5,'-o-offset10')
 text(100,10,'-d-offset20')
 text(100,9.5,'-^-offset20')
function [activeset, s0_d, s1_d,diff1,diff2,
theBER,theBER_o,pl,bs1ber,bs2ber]=
sho(k1,k2,d,r0,T_add,T_drop,timer,nUser1,nUser2,speed);
% IS-95 soft handoff algorithm
% sho(k1,k2,d,r0,T_add,T_drop,timer)
% Usage: [activeset s1 s2 diff1 diff2 theBER theBER_o
pl bs1ber bs2ber]=sho(0,30,2000,2000,-215,-217,10,10,10,10);
% approximate adding distance is 700m, drop at 1400m.
% k1 the BS transmitting power
% k2 the path loss parameter, usually 20-40
% d the distance between two BSs
% r0 the distance from BS0
% T_add adding threshold, -217 corresponds 600 meters for adding
% T_drop droping threshold, suggesting 1 db less than T_add
```

```
% timer hyperthesis time period
% speed meter/sec
%  u(d),v(d) is zero mean Gaussian process having the exponential
% correlation function as E(u(d)u(d+delta))=sigma^2*exp(delta/d0)
% d0 is a constant of decay. We assume sigma=8db, d0=20m
% ds is the sampling distance, use 1 meter here
d0=20;
ds=20;
sigma=12;
alpha=exp(-ds/d0);
sigmavi=(1-alpha^2)*sigma;
u=zeros(1,d);
v=zeros(1,d);
us=random('norm',0,sigmavi,1,d);
vs=random('norm',0,sigmavi,1,d);
u=us;
v=vs;
t0=0
t1=0
active(1)=1;
%% active(i)=1 BS1, =2 BS2, =3 BS1+BS2
for i=2 : d
s0_d(i)=k1-k2*log(i)+u(i);
s1_d(i)=k1-k2*log(d-i)+v(i);
diff1(i)=s0_d(i)-s1_d(i);
diff2(i)=s1_d(i)-s0_d(i);
active(i)=active(i-1);
```

```
if (s1_d(i)>T_add) & s0_d(i)>T_add,
active(i)=3;
end
if (s0_d(i)<T_drop) & active(i-1)==3 & (t0<timer ),
active(i)=3;
t0=t0+1;
end
if (s0_d(i)<T_drop) & active(i-1)==3 &(t0== timer),
active(i)=2;
t0=0;
end
if (s1_d(i)<T_drop) & active(i-1)==3 & (t1<timer ),
         active(i)=3;
         t1=t1+1;
end
if (s1_d(i)<T_drop) & active(i-1)==3 &(t1== timer),
         active(i)=1;
         t1=0;
end
%%BER theoretical BS1=5, BS2=10, powered by MaD.
[theBER(i,:),theBER_o(i,:),bs1ber(i,:),bs2ber(i,:)]=
   MaD(5,10,active(i),10^(s0_d(i)/10)/10^(s1_d(i)/10),
10^(s1_d(i)/10)/10^(s0_d(i)/10),0.3,i);
%%pl(1) relative signal strength in BS1 when power controled by BS2
%%pl(2) relative signal strength in BS2 when power controled by BS1
%SF1 vector for spreading factors for mobiles in BS1
%SF2 vector spreading factors for mobiles in BS2
```

```
SF1=ones(1,nUser1)*64;
SF2=ones(1,nUser2)*64;
pl(i,:)=[10^(s0_d(i)/10)/10^(s1_d(i)/10)
10^(s1_d(i)/10)/10^(s0_d(i)/10)];
%pl(i,:)=[s0_d(i)/s1_d(i) s1_d(i)/s0_d(i)];
link = [nUser1*100 SF1(1) nUser1];
physchan=[300 0]; % [SNR ];
multi=[1 0 0 0]; %multi path
misc=[1 1 0]; %reverse link

%[sum, BaseSignal, subsignal]=
cdma2(link,physchan,multi,misc,SF1,SF2,pl);
end

figure
plot([1:2000],theBER(:,2),'yo',[1:2000],theBER(:,3),'g+');
title('BER level using MaD scheme');
xlabel('mobile distance (meter)');
ylabel('BER theoretically');
text(10,0.005,'o SF=32\n + SF=64');
figure
plot(pl(800:1200,2))
xlabel('Mobile location in Handoff Area (meter)');
ylabel('Normalized Received Power Level in the Neighbor Cell');
figure
plot(bs1ber);
plot(bs2ber);
```

```
hold
plot([1:2000],0.2,'r-');
text(200,0.22,'desired BER for voice')
xlabel('Mobile location in Handoff Area (meter)');
ylabel('BER for voice traffic in the other cells');
figure
plot([1:2000],diff1,'yo',[1:2000],diff2,'g+')
xlabel('mobile distance in meter')
ylabel('signal strength in db')
text(100,diff1(100),'pilot signal from BS1')
text(1100,diff2(1100),'pilot signal from BS2')
activeset=active;
figure
plot(activeset)
xlabel('distance')
ylabel('active set')
axis([0 2000 0 4])
text(400,3.7,'active set, T_add=-15db, T_drop=-16db')
text(400,3.5,'T_tdrop=5 sec')
 figure
%%%theBER_of.eps
 hold
theBER_of=ones(2000);
theBER_of(700:1200)=theBER_o((700:1200),1);
theBER_of(1200:1300)=theBER_o((1200:1300),2);
plot([700:1300],theBER_of(700:1300));
plot([700:1300],0.02,'--')
```

```
text(850,0.02,'desired BER of mobile')
 xlabel('mobile distance')
 ylabel('BER')
 title('BER with Maximum Data rate and OR ON DOWN
  power control (towards high loaded cell)')
 text(1000,0.015,'SF=32')
 text(1200,0.007,'Cell 1 SF changed to 64')
line([800 1200],[0.007 0.007])
line([800 800],[0.006 0.009])
line([1200 1200],[0.006 0.009])
text(900,0.007,'handoff area, SF=32')
```

REFERENCES

[1] J. Wang, J. Liu, and Y. Lin. A partition-based parallel mpeg-2 software decoder. *Proc. of Joint Conference of Information Systems*, pages 1009–1012, Durham, North Carolina, Mar 2002.

[2] J. Wang and J. Liu. Parallel mpeg-2 decoding with high-speed networks. *Proc. of International Conference on Multimedia and Exposition 2001 (ICME 2001)*, pages 449–452, Tokyo, Japan, Mar 2001.

[3] J. Wang, Y. He, and J. Liu. Efficient buffering control for a software-only, high-level, high-profile mpeg-2 decoder. *to appear in IEEE International Conference on Multimedia and Exposition 2003 (ICME2003).*

[4] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. Rsvp: a new resource reservation protocol. *IEEE Network*, 7(5):8–18, Sep 1993.

[5] N. Kamat, J. Wang, and J. Liu. An efficient re-routing scheme for voice over ip. *to appear in IEEE International Conference on Multimedia and Exposition 2003 (ICME2003).*

[6] J.M. McManus and K.W. Ross. Video-on-demand over atm: Constant-rate transmission and transport. *IEEE Journal on Selected Areas in Communications*, 14(9):1087–1097, Aug 1996.

[7] A Durand. Deploying ipv6. *IEEE Internet Computing*, 5(1):79–81, Feb 2001.

[8] I. F. Akyildiz, J. McNair, L. Carrasco, R. Puigjaner, and Y. Yesha. Medium access control protocols for multimedia traffic in wireless networks. *IEEE Network*, pages 39–47, July/Augest 1999.

[9] H. Fukumasa, R. Kohno, and H. Imai. Design of pseudo-noise sequences with good odd and even correlation properties for ds/cdma. *IEEE Journal on Selected Areas in Communications*, 12(5):828–836, June 1994.

[10] ISO/IEC 13818-2. Information technology – generic coding of moving pictures and audio information: Video. 1998.

[11] T. Akiyama, H. Aono, R. Aoki, K.W. Ler, B. Wilson, T. Araki, T. Takahashi, H. Takeno, C. Boon, A. Sato, S. Nakatani, K. Horiike, and T. Senoh. Mpeg2 video codec using image compression dsp. *IEEE Trans. on Consumer Electron*, 40(3):466–472, Aug 1994.

[12] R. Lee. Realtime mpeg video via software decompression on a pa-risc processor. *40th IEEE Computer Society International Conference (COMPCON'95)*, pages 186–192, March 1995.

[13] R. Bhargava, L. John, B.L. Evans, and R. Radhakrishnan. Evaluating mmx technology using dsp and multimedia applications. *Proc. of the 31st IEEE International Symposium on Microarchitecture*, pages 37–46, 1998.

[14] A. Peleg, S. Wilkie, and U. Weiser. Intel mmx for multimedia pcs. *Communication of the ACM*, 40(1):25–40, Jan 1997.

[15] T. Tung, C. Ho, and J. Wu. Mmx-based dct and mc algorithms for real-time pure software mpeg decoding. *IEEE Multimedia Conference 99*, pages 357–362, July 1999.

[16] C.-G. Zhou, L. Hohn, D. Rice, I. Kabir, A. Jabbi, and X. Hu. Mpeg video decoding with the ultrasparc visual instruction set. *Compcon'95*, pages 470–475, Spring 1995.

[17] S. Sriram and C.-Y. Hung. Mpeg-2 video decoding on the tms320c6x dsp architecture. *Proc. of the 1998 IEEE Asilomar Conf. In Signal, Systems and Computers*, pages 1735–1739, 1998.

[18] I. Ahmad, S.M. Akramullah, M.L. Liou, and M. Kafil. A scalable offline mpeg-2 video encoding scheme using a multiprocessor system. *Parallel Computing*, 27(6):823–846, 2001.

[19] S. Akramullah, I. Ahmad, and M. Liou. A data-parallel approach for real-time mpeg-2 video encoding. *Journal of Parallel And Distributed Computing*, 30(2):129–146, Nov 1995.

[20] K. Gong and L. Rowe. Parallel mpeg-1 encoding. *Proc. of the 1994 Picture Coding Symposium*, Sep 1994.

[21] Y. He, I. Ahmad, and M. Liou. Real-time interactive mpeg-4 system encoder using a cluster of workstations. *IEEE Trans. on Multimedia*, 1(2):217–233, June 1999.

[22] A. Bilas, J. Fritts, and J. Singh. Real-time parallel mpeg-2 decoding in software. *Proc. of the 11th International Parallel Processing Symposium*, pages 37–46, 1997.

[23] N. H. C. Yung and K. K. Leung. Spatial and temporal data parallelization of the h.261 video coding algorithm. *IEEE Trans. on Circuits and Systems for Video Tech*, 11(1):91–104, Jan 2002.

[24] S. H. Bokhari. On the mapping problem. *IEEE Trans. on Computer*, c30(3):207–214, Mar 1981.

[25] N. Bowen and C. Nikolaou. On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems. *IEEE Trans. On Computer*, 41(3):257–273, Mar 1993.

[26] MPEG Software Simulation Group. Mpeg-2 video codec version 1.2. *http://www.mpeg.org/ tristan/MPEG/MSSG*, 04/25/2003.

[27] B. Daines, J. Liu, and K. Sivalingam. Supporting multimedia communication over a gigabit ethernet network. *International Journal of Parallel and Distributed Systems and Networks*, 4(2):102–115, Jun 2001.

[28] M. Naraghi-Pour and H. Liu. Integrated voice-data transmission in cdma packet pcn's. *Proc. of 2000 IEEE ICC*, pages 1085–1089, 2000.

[29] S. Choi and K. G. Shin. A unified wireless lan architecture for real-time and non-real-time communication services. *IEEE/ACM Trans. on Networking*, 8(1):44–59, 1999.

[30] L. Zhuge and V. Li. Interference estimation for admission control in multi-service ds-cdma cellular systems. *Proc. of 2000 IEEE GLOBECOM*, pages 1509–1514, 2000.

[31] M. Lops, G. Ricci, and A. M. Tulino. Narrow-band-interference suppression in multiuser cdma systems. *IEEE Trans. on Communications*, 46(9):1163–1175, 1998.

[32] P. Chang and C. Lin. Design of spread spectrum multi-code cdma transport architecture for multimedia services. *IEEE Journal on Selected Areas in Communications*, 18(1):99–111, January 2000.

[33] E. Geraniotis and J. Wu. The probability of multiple correct packet receptions in direct-sequence spread-spectrum networks. *IEEE Journal on Selected Areas in Communications*, 12(5):871–884, June 1994.

[34] S. Choi and D. Cho. Capacity evaluation of forward link in a cdma system supporting high data-rate service. *Proc. of 2000 IEEE GLOBECOM*, pages 123–127, 2000.

[35] I. F. Akyildiz, D. A. Levine, and I. Joe. A slotted cdma protocol with ber scheduling for wireless multimedia networks. *IEEE/ACM Trans. on Networking*, 7(2):146–157, April 1999.

[36] S. Oh and K. M. Wasserman. Dynamic spreading gain control in multi-service cdma networks. *IEEE Journal on Selected Areas in Communications*, 17(5):918–927, May 1999.

[37] M. Elicin, J. Wang, and J. Liu. Support multimedia traffic over w-cdma with dynamic spreading. *Proc. of GLOBECOM'2001*, 4:2384–2388, 2001.

[38] J. Wang, M. Elicin, and J. Liu. Multimedia support for wireless w-cdma with dynamic spreading. *ACM Journal of Wireless Network*, 8:355–370, 2002.

[39] J. Wang, Y. Cen, and J. Liu. Analysis of soft handoff algorithms with dynamic spreading wcdma system. *Proc. of the 17th International Conference on Advanced Information Networking and Applications (AINA2003)*, 2003.

[40] J. Wang, J. Liu, and Yuehao Cen. Handoff algorithms in dynamic spreading wcdma system supporting multimedia traffic. *IEEE Journal on Selected Areas in Communications*, to appear in 2003.

[41] K.S. Gilhousen A. J. Viterbi, A.M. Viterbi and E. Zehavi. Soft handof extends cdma cell coverage and increases reverse link capacity. *IEEE Journal on Selected Areas in Communications*, 12:1281–1288, Oct 1994.

[42] N. Zhang and J.M. Holtzman. Analysis of a cdma soft handoff algorithm. *IEEE Trans. on Vehicular Technology*, 47(2):710–714, May 1998.

[43] Y. Fang and I. Chlamtac. Analytical generalized results for the handoff probability in wireless networks. *IEEE Trans. on Communications*, 50(3):396–399, March 2002.

[44] S. Chakrabarti P. Marichamy and S.L. Maskara. Overview of handoff schemes in cellular mobile networks and their comparative performance evaluation. *Proc. of IEEE VTC'99*, 3:1486–1490, 1999.

[45] C.C Lee and Raymond Steele. Effect of soft and softer handoffs on cdma system capacity. *IEEE Trans. on Vehicular Technology*, 47(3):830–841, Aug, 1998.

[46] C. Zhou, L. Honig, and S. Jordan. Two-cell utility-based resource allocation for a cdma voice service. *IEEE Proc. of 54th Vehicular Technology Conference*, 1:27–31, 2001.

[47] M. Naghshineh and M. Schwartz. Distributed call admission control in mobile/wireless networks. *IEEE Journal of Selected Areas on Communication*, 14(4):711–717, May 1996.

[48] T. Lee and J. Wang. Admission control for variable spreading gain cdma wireless packet networks. *IEEE Trans. on Vehicular Technology*, 49(2):565–575, March 2000.

[49] T. Chiang and Y.-Q Zhang. A new rate control scheme using quadratic rate distortion model. *IEEE Trans. on Circuits and Systems for Video Technology*, 7(1):246–250, Feb 1997.

[50] W. Ding and B. Liu. Rate control of mpeg video coding and recording by rate- quantization modeling. *IEEE Trans. on Circuits and Systems for Video Technology*, 6(1):12–20, Feb 1996.

[51] H. Hang and J. Chen. Source model for transform video coder and its application—part i: Fundamental theory. *IEEE Trans. on Circuits and Systems for Video Technology*, 7(2):287–298, April 1997.

[52] H. Hang and J. Chen. Source model for transform video coder and its application—part ii: Variable frame rate coding. *IEEE Trans. on Circuits and Systems for Video Technology*, 7(2):299–311, April 1997.

BIOGRAPHICAL SKETCH

Ju Wang was born in Feng Hua, P. R. China. He received his Bachelor of Science degree from the Computer Science Department at Ocean University of Qingdao, P. R. China in 1995. In 1998, he earned his Master of Science degree from the Computer Science Department at Institute of Computing Technology, Chinese Academy of Sciences, P. R. China. He is expected to receive his Doctor of Philosophy degree in computer and information science and engineering from University of Florida, in August 2003. He has accepted an assistant professor position with Virginia Commonwealth University, starting Augest 2003. His research interests include distributed multimedia systems, digital video processing, computer networking, wireless networks and security.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Jonathan C.L. Liu, Chairman
Associate Professor of Computer and Information Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Randy Y. C. Chow
Professor of Computer and Information Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Jih-kwon Peir
Associate Professor of Computer and Information Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Sartaj Sahni
Professor of Computer and Information Science and Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Yuguang "Michael" Fang
Assistant Professor of Electrical and Computer Engineering

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 2003

Dean, College of Engineering

Dean, Graduate School