CCE 4001 – Advanced Computer Architecture

Assignment

A Framework for Heuristic Scheduling for Parallel Processing on Multicore Architecture: A Case Study with Multiview Video Coding

The paper attached deals with a multicore solution for multiview video coding.

(a) Discuss the contents of this paper, highlighting the advantages and disadvantages of the proposed architecture.

(70% of marks)

(b) Suggest methods to enhance the efficiency of this architecture and single out any defects that you think are incorporated within this design.

(30% of marks)

Use any published material to sustain your arguments.

The submitted report should follow A4 IEEE double column format with single-spaced, twelve-point font in the text. The maximum report length is four (4) pages. Reports in excess of four pages will not be read and a zero mark will be assigned. All figures, tables, references, etc. are included in the page limit. A template in Word or Latex can be downloaded from http://www.ieee.org/go/conferencepublishing/templates.

Hard deadline for the submission of the assignment: 15th January 2010 at 12:00. No Assignment will be accepted after this date and time. Students can work in a group, but each group is limited to a maximum of two.

Dr Ing. Carl James Debono
A Framework for Heuristic Scheduling for Parallel Processing on Multicore Architecture:
A Case Study with Multiview Video Coding

Yi Pang, Student Member, IEEE, Lifeng Sun, Member, IEEE, Jiangtao (Gene) Wen, Senior Member, IEEE,
Fengyan Zhang, Weidong Hu, Wei Feng, and Shiqiang Yang, Senior Member, IEEE

Abstract—In this paper, using the Intel multicore architectures and the emerging multiview video coding standard, we introduce a framework for performing analysis, simulation, and evaluation of heuristics scheduling algorithms for implementing computationally intensive algorithms on multicore processors. The framework allows for accurate and quantitative characterization of the performance of dynamic scheduling algorithms for multimedia applications on different multicore processors without actual implementation of the scheduling algorithm and application on the actual platform. Experimental results demonstrate the effectiveness and scalability of our framework.

Index Terms—Directed acyclic graph, multicore architecture, scheduling algorithms.

I. INTRODUCTION

IN SPITE OF the rapid improvement of microprocessor technology, many applications remain outside of the realm of practicality on even the most cutting-edge microprocessor platforms. One such computationally intensive application is video encoding, which has evolved in the last two decades from MPEG-2 to advanced video coding (AVC), as well as multiview video coding (MVC) and scalable video coding. Compared with MPEG-2, currently the most widely deployed video coding standard in the world, AVC delivers twice the video coding efficiency at significantly higher computation requirements. This situation is illustrated in Fig. 1, which shows the rate of increase for the relative computation complexity of different video coding standards, as well as the rate of growth in computational capabilities for uncore and multicore microprocessors over the same period of time. The rates of growth in both cases are normalized to the year 1995, when MPEG-2 was first widely adopted.

A recent trend in microprocessor design is to put multiple processing cores on the same processor. The presence of multiple cores not only improves the amount of data that the processor can handle in parallel, but also reduces the cost of data communications and the bandwidth required for data exchange prior to/after processing between the cores, which in turn reduces bandwidth requirement for the communication between the processor and off-chip memory, as well the power consumption of the entire chip. Asanovic et al. [1] expresses the landscape of parallel computing research.

The challenge, however, in utilizing multicore technology for computation and data intensive applications such as video coding, is to “map” the application onto a multicore processor, so that the cores are fully utilized and unnecessary data exchanges between the cores and especially between the processor and external, off-chip memory are eliminated to the greatest extent.

Most computation and data intensive applications such as media processing, involves computation intensive tasks that can be parallelized. The frequency and scope of these operations, however, are usually context dependent, which makes it difficult for a static scheduling algorithm to achieve optimized performance partitioning such applications onto multicore processors. As an example, in the baseline and main profiles of the AVC standard, the motion prediction residual of each 16 × 16 macroblock (MB) is partitioned into nonoverlapping 4 × 4 blocks that subsequently undergo 4 × 4 integer transform, quantization, and entropy coding. Although the transform of each 4 × 4 block is independent of other 4 × 4 blocks, the entropy coding of the coefficients after quantization is context dependent, and must be processed sequentially. Therefore, even if the transforms of the 4 × 4 blocks in an MB is partitioned onto different processing cores, the resultant coefficients must still be exchanged between the cores so that entropy coding could be carried out correctly. In
In this paper, we use the emerging MVC standard as an example and present a generic directed acyclic graph (DAG)-based framework that can be used to model the dynamic nature of the dependencies between tasks in a complicated application at different granularities. We demonstrate ways of using such a framework to come up with different heuristics for designing efficient scheduling algorithms, and more importantly, ways of evaluating or predicting the performance of scheduling algorithms for applications on multicore platforms without actually implementing the algorithm and the application on the target platform. In the paper, we use five different and commonly adopted heuristic-based scheduling algorithms for MVC with different encoding options as test cases. Through both analysis and experimental results, we show that the framework, tools, and methodology presented in this paper is capable of capturing the key characteristics and performance parameters of a scheduling algorithm.

The paper is organized as follows. Section II contains an analysis of the problem of scheduling video encoding tasks on multicore architectures. In Section III, we introduce the DAG-based framework, as well as methodologies for using such a framework for the design of heuristic for finding good scheduling algorithms and to analyze their performances. Using MVC as an illustrative example, in Section IV, we demonstrate the framework in operation and its usefulness, as well as experimental results. Finally, Section V contains the conclusions and suggestions for future research.

II. Parallel Scheduling of Video Encoding for Multicore Architecture

Video encoding is well known to be computationally demanding, and therefore, parallel architecture for video encoding has been an active research topic for well over a decade, and will continue to be so in the foreseeable future. Interested readers may refer to references [2]–[5]. In [6], Wang et al. proposed a novel high-definition television video decoder and decentralized control scheme with a data-driven architecture on the function level. The data-driven architecture was adapted to allow each processing unit to operate as soon as data and buffer become available. In [7], Nanda et al. introduced IBM’s video surveillance server prototype, implementing the MVC on the cell broadband engine (Cell/B.E.) processor. The tasks are partitioned into four modules, each of which is assigned to a dedicated synergistic processing element.

In addition to single view coding, MVC as an extension to AVC was recently standardized to efficiently compress multiple parallel video inputs that capture different views of the same content. Although it is intuitive to see that MVC exhibits good inherent parallelism, tackling the encoding
and decoding of multiple views, requires higher processing power, more storage, and higher bandwidth communications. As a result, implementing MVC on multicore processors has become an important emerging research topic. In [8], a parallelization methodology for MVC based on hyper-space theory was presented and tested on multiprocessor platforms. [9] analyzed the parallelism of MVC using video frames as the parallel granularity. The techniques and methodology in these papers differ significantly as a result of the differences in target applications and platforms, which highlighted the need for a systematic and generic approach for performing theoretical analysis and design of scheduling and parallelization algorithms for video encoding on multicore processors.

A fundamental tool that we are using throughout this paper is DAG. Scheduling algorithm design and analysis for parallel systems and architectures using DAGs is a classic problem (see for example, [10]–[12]). This prior research, however, was not targeted at scheduling dynamic tasks on multicore architectures, but toward cluster, grid, and multiprocessor systems that involves tradeoffs that are different than MVC. These existing approaches targeted cases in which the DAGs are to a very large extent static, as opposed to involving dynamic insertion and deletion of nodes over time. Further more, the application of the DAG-based approach is mainly to facilitate only the designing of scheduling algorithms, as opposed to designing and then evaluating such algorithms, without actually implementing the scheduling algorithm in question.

In this paper, without losing generality, we use the total execution time as the criterion for measuring system performance, although other criteria are also possible.

In traditional sequential implementations of video encoding on a unicore processor, at each given moment, only one group of pictures (GOPs) can be processed and the encoding of the next GOP can not start until the current GOP is completed. In MVC and on multicore platforms, however, GOPs can be processed in parallel by the different cores. The number of GOPs that a multicore processor can process simultaneously is called the number of concurrent GOPs (NoC GOPs), which is limited by the delay constraints of the application and the sizes of the encoding and decoding buffers. When a processing core of the system is done with encoding a GOP of the input, it will “bypass” the other GOPs that the other cores are processing, and may go directly to the next GOP that has not been processed by any of the cores. In practice and to simplify data access strides, the allocation of the GOPs to be encoded by each core is usually periodic, as shown in Fig. 2 for the cases when the NoC GOPs is 1 or 3. The arrows in the figure illustrate the time order in which the GOPs are processed, and the interleaving of arrows depict the concurrent nature of the allocation of GOPs to cores.

Because the data dependencies in video encoding in general and MVC in particular, are complex, time variant, and sometimes dependent on the “earlier” decisions made by a dynamic scheduling algorithm, finding the optimal scheduling algorithm for MVC with a certain configurations (e.g., number of views, resolution, bit-rate, content, etc.) is often nondeterministic polynomial complete or at least immensely computational intensive, thereby making it difficult for embedding into real-time encoding operations for improving system performance “on-the-fly.”

III. HEURISTIC PARALLEL SCHEDULING FRAMEWORK OF VIDEO ENCODING ON MULTICORE ARCHITECTURE

A. Problem Formulation

To model video encoding on a certain multicore processor, we first introduce a set of variables A which uniquely identifies a configuration of a particular encoding application on a particular multicore processor, e.g., MVC with eight views and other encoding parameters on IBM Cell/B.E. with eight cores, etc. The subset of A that is related to video encoding and impacts compression efficiency (Y) is denoted by A_Y. Obviously, Y is a function of A_Y, i.e., Y = f(A_Y).

A particular video encoding application is characterized by a series of encoding tasks that may change over time. Depending on the particular application, the tasks could be defined at different granularities, e.g., encoding a GOP, a frame, a slice, so on, and so forth. When the ith task is processed, the next scheduling action f(i) is constrained by the application and the platform, as well as earlier scheduling decisions. We denote the time between the completion of the i - 1th task and the ith task by , It is straightforward to see that the total execution time t(i) from beginning encoding to the completion of the ith task is simply t(i) = \sum_{j=1}^{i} \Delta t(j). Given set A and a scheduling algorithm, the total execution time t(n - 1) is uniquely decided, as in T(n - 1) = \sum_{j=0}^{n-1} \Delta t(j) \sum_{i=0}^{n-1} t(j), j = 0, 1, \ldots, |Q| - 1 where, n is the number of tasks, T is the set of all possible total execution times and Q is the set of heuristics scheduling algorithms. Usually, different heuristics lead to different scheduling algorithms. The problem of finding the optimal scheduling algorithm can be formulated as

\[
\min_{Q \in T(n - 1)} \sum_{i=1}^{n-1} \Delta t(i) + \sum_{j=0}^{n-1} t(j), \quad j = 0, 1, \ldots, |Q| - 1
\]

s.t. \(Y(A_Y) \geq Y_0\) (1)

where \(Y_0\) is the compression performance requirement of the application (e.g., as defined by target bit-rate/peak signal-to-noise ratio).

In the above formulation, the heuristics scheduling algorithm that leads to the shortest total execution time is considered optimal. As noted earlier, it is straightforward to generalize the formulation to other optimality criteria. It should be noted that we assume that the video encoder is causal and 1-pass, i.e., it is not possible to revert to a decision that the encoder and/or the scheduler has made earlier. This is a realistic assumption for most real-time video applications.

B. Application to MVC

There are different levels of the syntax elements in MVC that can be used as parallel granularities, including: GOP, frame, slice, MB, etc. MVC also consists of different encoding steps, including intra-prediction (Intra-P), inter-prediction (Inter-P), residual coding, motion estimation, quantization, etc. The subset of \(A\) that is related to video encoding and impacts compression efficiency (\(Y\)) is denoted by \(A_Y\).

One approach to video encoding on multicore processors is to consider the encoding of frames as a DAG where the vertices represent the encoding steps and the edges represent the data dependencies that must be satisfied. The DAG is then scheduled on the multicore platform to minimize the total execution time. This approach can be used to achieve high parallelism and to take advantage of the parallel processing capabilities of the multicore platform.

However, this approach has some limitations. First, it assumes that the DAG can be accurately modeled, which may not be the case in practice. Second, it may not be possible to exploit all the parallelism in the DAG, especially for larger video sequences. Third, the scheduling algorithm may not be able to find the optimal schedule, which may result in suboptimal performance.

To overcome these limitations, we propose a heuristic scheduling framework for video encoding on multicore processors. This framework is based on the parallelization methodology described in Section II and the problem formulation presented in Section III. The framework consists of two main components: (1) a DAG-based approach for video encoding that is used to model the encoding process, and (2) a scheduling algorithm that is used to determine the order in which the encoding tasks are executed.

The DAG-based approach models the video encoding process as a DAG where the vertices represent the encoding steps and the edges represent the data dependencies that must be satisfied. The DAG is then scheduled on the multicore platform to minimize the total execution time. This approach can be used to achieve high parallelism and to take advantage of the parallel processing capabilities of the multicore platform. However, this approach has some limitations, as discussed above.

To overcome these limitations, we propose a heuristic scheduling framework for video encoding on multicore processors. This framework is based on the parallelization methodology described in Section II and the problem formulation presented in Section III. The framework consists of two main components: (1) a DAG-based approach for video encoding that is used to model the encoding process, and (2) a scheduling algorithm that is used to determine the order in which the encoding tasks are executed.

The scheduling algorithm is based on a set of heuristics that are designed to find a near-optimal schedule. The heuristics are based on the total execution time of the tasks and the dependencies between the tasks. The scheduling algorithm is implemented as a heuristic parallel scheduling framework that is able to schedule the tasks in a way that minimizes the total execution time. The framework is designed to be scalable and able to handle large video sequences.

The scheduling algorithm is implemented as a heuristic parallel scheduling framework that is able to schedule the tasks in a way that minimizes the total execution time. The framework is designed to be scalable and able to handle large video sequences. The framework is implemented as a set of heuristics that are designed to find a near-optimal schedule. The heuristics are based on the total execution time of the tasks and the dependencies between the tasks. The scheduling algorithm is implemented as a heuristic parallel scheduling framework that is able to schedule the tasks in a way that minimizes the total execution time. The framework is designed to be scalable and able to handle large video sequences. The framework is implemented as a set of heuristics that are designed to find a near-optimal schedule. The heuristics are based on the total execution time of the tasks and the dependencies between the tasks.
dependencies, and reflects the constraints and “cost” structure imposed by the target platform, and traversing through this DG reflects the performances of a particular scheduling algorithm.

C. Heuristics Algorithm Design Using DAG—Overview

We introduced two types of graphs in the previous section; one models the dynamic dependencies between encoding tasks among encoding units, and the other models the capabilities of the multicore platform.

To find a good heuristic algorithm \( q \) using these two graphs, we introduce another set of parameters \( X \), which quantitatively describe the performance of the scheduling algorithm. Typical members of \( X \) may include the utilization of each core, the ratio of executing time to data transmission time, etc.

Similarly, heuristics algorithm can also be parameterized by a vector \( Z \), whose components may include parallel granularity, heuristics mechanism, etc. \( Z \) is a function of \( X \), and each \( Z \) corresponds to a different \( q \).

For example, when the utilization of core is low, a heuristics scheduling algorithm \( q \) may aim to generate a lower parallel granularity. To this end, \( q \) can lower the parallel granularity and choose to schedule a task that is prerequisite for the most of other tasks. If multiple tasks have the same dependent tasks, then the task with the shortest executing time should be scheduled.

On the other hand, when the ratio of executing time to data transmission is low, the goal of \( q \) maybe to reduce data transmission. To achieve this goal, \( q \) can schedule the tasks that are self-contained to each core thereby reducing data transmission.

D. Framework Design Using DAG

In this section, we demonstrate the design of heuristics mechanisms \( q \) using the DAG-based framework presented earlier. The framework (shown as Fig. 5) consists of three modules: the encoding module, the heuristics scheduling module, and the multicore module.

Encoder Module: Video encoding algorithm is abstracted as a hierarchical multisource multisink DAG extended over the time axis. The DAG \( G \) is updated continuously to reflect the input that is currently “visible” to the encoder (e.g., the current frame and the frames in the reference frame buffer).

The main task of this module is to update the DAG once a task has been scheduled. \( G(i) \) denotes the updated graph \( G \) after scheduling the \( i \)th task.

<table>
<thead>
<tr>
<th>Granularity</th>
<th>GOP Level</th>
<th>Frame Level</th>
<th>MB Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MB</td>
<td>Concurrent GOP</td>
<td>Intra-P</td>
<td>Null</td>
</tr>
<tr>
<td>Frame</td>
<td>Concurrent GOP</td>
<td>Intra-P</td>
<td>Null</td>
</tr>
<tr>
<td>GOP</td>
<td>Concurrent GOP</td>
<td>Null</td>
<td>Null</td>
</tr>
</tbody>
</table>

Fig. 3. Hierarchical dependence of MVC—an example.

Fig. 4. DAG of MVC with two concurrent GOPs.
Heuristic Scheduling Module: The heuristics scheduling module synthesizes the states of encoder graph and multicore processor, and makes task scheduling decisions denoted by $f(i)$. Each heuristics mechanism corresponds to a particular scheduling algorithm.

Multicore Module: The module records and updates the states of the multicore processor, and calculates the set $\{\text{states of nodes}\}$ of tasks which have been finished on the $n$th step. Updates to the module occur only when $i$ is updated to $i+1$.

Conceptually, the work flow is as the following.

1) Initialization: Initialize parameters and states of. Set $i = 0$, $G(i) = G(0)$, where $G(0)$ is the initial $g$ (the number of concurrent GOPs) GOPs. The states of cores and data transmission bandwidths are set free.

2) Scheduling: Based on the states of cores and data transmission and $G(i)$, $q$ chooses the next scheduling action $f(i)$ based on both video encoding and processor constraints.

3) Update multicore states: When a task is scheduled, the states of the processor must be updated.

4) Calculate the set $V(i)$ of nodes which are finished: When a round of tasks for all the cores has been completely scheduled, the virtual time is incremented and the set $V(i)$ is null. Here, the term “virtual time” refers to the total execution time as captured by the simulation framework (as opposed to measured on the actual platform).

5) Update graph and simplify graph $G$. When new nodes in current round have been encoded, i.e., $V(i)$ is not empty, graph $G$ needs to be updated to $G(i+1)$ referring $V(i)$. If $V(i)$ is empty, $G(i+1) = G(i)$.

6) Repeat steps 2-5 until all frames are encoded.

Fig. 5. Structure diagram of framework.

Fig. 6. GOP prediction structure in MVC.

IV. APPLICATION TO MVC ON SYMMETRIC MULTICORE ARCHITECTURES

To further illustrate the DAG-based approach, in this section, we use the scheduling of MVC on the Intel symmetric multicore processor as an example.

A. Framework for MVC

1) Encoder Module for MVC: In this particular example, we assume that the processing capability of each of the symmetric cores is $c$. This is a reasonable assumption because for each frame, the execution time of ME, DCT, quantization, and entropy coding is much higher than that of data transfer on symmetrical multicore architecture with shared memory, such as the Intel multicore processor. Under these two assumptions, we obtain $W/c$ through experiment and categorize the tasks according to their, respectively, times. Through experiments we found that the ratio of the execution times for different GOPs remain relatively constant for different clips, even though the value of the execution time itself may change.

During initialization, $G(0)$ is initialized to $g$ connected GOPs. As defined earlier, $G$ is the set of edges of the encoder DAG. $E$ is the set of edges of the encoder DAG. We use $\omega$ to denote the number of GOPs in the input.

Under these assumptions, pseudo code of the update procedure $UpdateG(i)$ for $G$ is shown in $UpdateG(i)$. In our experiments we chose frame as the parallel granularity, while GOP and MB are also possible choices. Through experiments we categorize the frames into five classes, namely I, P, B2, B3, and B4. I and P types correspond I-frame and P-frame, respectively.GOPs, $\omega(n = 2, 3, 4)$ type corresponds B-frame which refers to $n$ frames. The estimated execution time of the node is estimated-based on the frame type. The most effective way of setting the weights is to based the value on the ratio of the time the processor needs for performing different tasks.
The ratio usually is relatively easier to obtain. In the paper, for example, to establish the framework for MVC, we need to know the ratio of running times for encoding I-frame, P-frame and B-frames (n = 2, 3, 4). Such data can be estimated using virtually any implementation of MVC on the target platform. Fig. 7 shows the scheduling of a GOP in MVC. The scheduling (Fig. 7) can be generated from Fig. 6 automatically by topological sorting. The dependences in Fig. 6 remain in Fig. 7. In order not to make Fig. 7 too complex, we eliminated some nonessential edges, e.g., edge from frame 00 to 20, the edges from 00 to 40 and from 40 to 20 imply that frame 20 depends on 00.

2) Heuristics Scheduling Module for MVC: In our experiments, we tested several different heuristic scheduling algorithms: Poc, Refs. Num., Ex. Time, Sum. Time, and Left Frame, which respectively gives the highest priority to: 1) the earliest frame in display order; 2) the frame which is referred by the most frames; 3) the frame with the shortest expected execution time; 4) on the subsequent route in G, the frame whose sum of executing time is the largest; and 5) on the subsequent route in G, the frame which is referred by the most frames. Task pool TP(i) is the set of nodes which are ready to process in graph G. The task which has the highest priority is scheduled first. The scheduling action of the ith task to the u(i)/th core is denoted by f(i) = (f(u(i), u(i))), i = 0, 1, ..., n − 1, in which f(u(i)) is the serial number of task.

3) Multicore Module for MVC: Because we assume that the cost of communication is negligible, the multicore module simply records and updates Sours = {core0, core1, ..., corei−1}. The states of the cores is reflected in terms of remaining time for the task. We assume corei = (cti, cnj), j = 0, 1, ..., m − 1, ctj denotes the left time of task scheduled on the core. If ctj = 0, the corresponding core is free. cnj denotes the serial number of task scheduled on the core, τnj is the

![Fig. 7. GOP scheduling in MVC.](image)

![Fig. 8. Working flowchart of framework.](image)
We then tested the standard golf1 sequence (320 × 240) [14] on the real system with the same parameters as Figs. 10 and 12. As can be seen from the figures, the actual performance on the actual system is nearly identical to the prediction in Figs. 9 and 11. This comparison shows that our framework can be used as a tool for reliably evaluating the performance of different scheduling algorithms without implementing the application and the scheduling algorithm on the actual platform.

Figs. 13 and 14 show the speedup of Poc encoder with 2, 3, 4 and an unlimited number of concurrent GOPs, as analyzed using the proposed framework and measured on the actual system, respectively. Again, the prediction and the actual measurement matches nicely. We can also see that as the NoC GOP increases, the speedup also increases. When the number of concurrent GOPs is unlimited, the slope of the speedup of dynamic heuristics algorithms is nearly linear.

We also tested other standard test sequences including Rena (640 × 480) [15], Ballet (1024 × 768) and Breakdancer (1024 × 768). The results are shown in Fig. 15.

It is clear from Fig. 15 that proposed framework and algorithms perform for these additional test sequences are similar to that of golf1, the speedup is linear with respect to the number of cores and the analysis with the framework matches the actual measurement nicely.

In summary, the comparison of the framework and performance on the real system shows that our framework can be used to predict and analyze critical characteristics of the performance of an algorithm for an application on the actual
V. CONCLUSION AND FUTURE RESEARCH

The increased interests in the design and adaption of a multicore platform for computationally intensive and dynamic applications such as MVCs necessitate dynamic scheduling algorithms and tools for analysis and fine tuning such algorithms, and for predicting important characteristics of the performance (e.g., core usage, speedup ratio, bandwidth, etc.). In this paper, a framework for modeling the behavior of the target application and target platform using DAGs was introduced, with simulation results demonstrating the usefulness and scalability of the framework and methodology presented. The framework and tools introduced in this paper are generic, expandable to other applications and optimality metrics, and lightweight. It can be used to accurately capture the performance of applications and algorithms that would otherwise take much longer to implement, debug, and test, and/or for platforms that are not yet available for actual implementation.

In our experiments for this paper, the scheduling algorithms that we tested for each test scenario are static, i.e., even though input to the algorithm maybe highly dynamic, the criteria used by the algorithm for tasking scheduling (e.g., Poc or otherwise) remains unchanged for the entire application.

Because of the dynamic and lightweight real-time nature of the tools that we introduced, a perceivable and important area for future research is embedding the tools introduced in the paper into the application with feedbacks from the physical platform (e.g., core utilization, etc.) incorporated into the decision loop so that a truly dynamic and real-time optimized scheduling can be achieved.

ACKNOWLEDGMENT

The authors would like to thank Intel China Research Center for providing an experimental environment.

REFERENCES


Yi Pang (S’07) received the B.S. and M.S. degrees in computer science from Tsinghua University, Beijing, China, in 2005 and 2007, respectively. She is currently pursuing the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University.

During her B.S. and Ph.D. study, she held several internships: Summer Senior Analyst at Lehman Brothers, Hong Kong; a Software Engineer at Bitnaraiz, Munich, Germany, an Instructor at Qinghua University, Xining, China, etc. Her research interests include parallelism of multimedia (e.g., multiview video coding, H.264, ray tracing, etc.) on multicore processors (e.g., Cell/B.E., Intel multi-core, etc.).

Lifeng Sun (M’05) received the B.S., M.S., and Ph.D. degrees in system engineering from the National University of Defense Technology, Hunan, China, in 1995, 1997, and 2000, respectively. He is currently an Associate Professor in the Department of Computer Science and Technology, Tsinghua University. During 1998-1999, he was a visiting scholar at the University of California, Los Angeles (UCLA), where he conducted cutting-edge research on multimedia coding and communications. Many of his inventions there were later adopted by international standards such as H.263, MPEG, and H.264. After UCLA, he served as the Principal Scientist at PacketVideo Corp., the Chief Technical Officer at Morphisus Technology Inc., the Director of Video Coding Technologies at Mobileye Corp., and as a Technology Advisor at Ottera Wireless and Strong, Inc. Since 2009, he has been a Professor at the Department of Computer Science and Technology, Tsinghua University. He is a world-renowned expert in multimedia communication over hostile networks, video coding, and communications. He has authored many widely referenced papers in related fields. Products deploying technologies that he developed are currently widely used worldwide. He holds over 30 patents with numerous others pending.

Mr. Yang is currently the President of the Multimedia Committee of the China Computer Federations.

Fengyan Zhang was born in 1983. She received the B.S. degree in computer science and technology from Tsinghua University, Beijing, China, in 2006. She is currently pursuing the M.S. degree in computer science and technology from the Department of Computer Science and Technology, Tsinghua University.

Her current research interests include parallel video coding and multiview video coding, among others.

Wei Dong Hu was born in 1986. He received the B.S. degree in computer science and technology from Tsinghua University, Beijing, China, in 2009. He is currently pursuing the M.S. and Ph.D. degrees in computer science and technology from the Department of Computer Science and Technology, Tsinghua University.

His current research interests include parallelism of multimedia on multicore processors.

Mr. Hu won the gold medal from the Association for Computing Machinery International Collegiate Programming Contest in March 2007, and the gold medal in the International Olympiad in Informatics (IOI) in 2004 and in 2005. He is the coach of the IOI China Team.

Wei Feng was born in 1987. He is currently pursuing the B.S. degree in computer science and technology from Tsinghua University, Beijing, China.

His research interests include high-performance computing, parallel video coding, and multiview video coding, among others.

Shiqiang Yang (M’97–SM’08) graduated from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 1977, and received the M.E. degree in 1983.

He is now a Professor at Tsinghua University. His research interests include multimedia technology and systems, video compression and streaming, content-based retrieval, semantics for multimedia information, and embedded multimedia systems. He has published more than 100 papers in international conferences and journals.

Mr. Yang is currently the President of the Multimedia Committee of the China Computer Federations.