Design and Analysis of MPEG-2 MP@HL Decoder in Multi-Processor Environments

Seunghwan Yoo[1], *Hyun-Seung Lee*[1], *Sang-Jo Lee*[2], *Rae-Hong Park*[1], *and Do-Hyung Kim*[2]

[1]Department of Electronic Engineering
Sogang University
Seoul, Korea
E-mail: {adri00; billy78; rhpark}@sogang.ac.kr

[2]Samsung Advanced Institute of Technology (SAIT)
Suwon, Korea
E-mail: {sjlee92; hyungk}@samsung.com

ABSTRACT

As demands for high-definition television (HDTV) increase, the implementation of real-time decoding of high-definition (HD) video becomes an important issue. The data size for HD video is so large that real-time processing of the data is difficult to implement, especially with software. In order to implement a fast moving picture expert group-2 decoder for HDTV, we compose five scenarios that use parallel processing techniques such as data decomposition, task decomposition, and pipelining. Assuming the multi digital signal processor environments, we analyze each scenario in three aspects: decoding speed, L1 memory size, and bandwidth. By comparing the scenarios, we decide the most suitable cases for different situations. We simulate the scenarios in the dual-core and dual-central processing unit environment by using OpenMP and analyze the simulation results.

Keywords: HD video, MPEG-2, MP@HL decoder, multi-processor, parallel processing

1. INTRODUCTION

With the development of display technology, demands for high-definition (HD) video have increased. Because the amount of HD video data is tremendous, the processing of HD video must be fast and efficient for real-time processing. The amount of raw data to be processed per second is about 750 Mbits for the HD video with the following specifications: pixel resolution of 1920 by 1080 with 4:2:0 color format at the frame rate of 30 fps. Moving picture expert group (MPEG)-2 main profile at high level (MP@HL) is chosen as standard codec for HD television (HDTV) [1], and various methods have been proposed and developed for fast and efficient encoding/decoding process for HDTV [2]-[8].

Encoding process does not have to be of real-time, but decoding process should be fast enough to be performed in real-time. For HD video decoding, both hardware (HW)- and software (SW)-based methods can be used. Dedicated HW solutions are fast enough to be of real-time, but its cost is high and it is not flexible. On the contrary, SW-based solutions have many advantages [9]: they provide flexibility without any additional expenses, and run on general-purpose systems. Thus, in many cases, SW-based methods are preferred. However, since they are slower, it is a challenging job to decode HD video in real-time. To overcome these drawbacks, parallel computation techniques can be employed. Many researches have proposed multi-processing architectures for encoding/ decoding processes [9]-[12].

In this paper, we found an efficient way to use the parallelism for MPEG-2 MP@HL decoder. First, we find the feasible combinations of parallel computing techniques: data decomposition, task decomposition, and data-flow decomposition. Assuming the multi-digital signal processor (DSP) environments, we analyze each scenario in terms of the decoding speed, cache memory size, and bus bandwidth. Simulation results in the dual-core and dual-central processing unit (CPU) personal computer (PC) environment are shown and compared with the assumed multi-DSP environment.

The rest of the paper is structured as follows. First, Section 2 presents an overview of MPEG-2 decoder system. Section 3 focuses on fundamentals of parallel processing and its implementation, and Section 4 describes five scenarios of parallel MPEG-2 MP@HL decoder. We analyze simulation results in terms of the decoding speed, cache memory size, and bus bandwidth in Section 5. Finally, conclusions are given in Section 6.

2. MP@HL MPEG-2 DECODER

Prior to designing a parallel MPEG-2 decoder for HD video, we need to overview the MPEG-2 video decoder and modularize by task partition.

2.1 Overview of MPEG-2 Decoder

MPEG-2, which is the enhanced version of MPEG-1, is one of the standards for video and audio codec. In this paper, we focus on the Part 2 (Video) of MPEG-2. Decoding process of MPEG-2 video is the inverse of the encoding process. Fig. 1(a) shows the block diagram of an MPEG-2 decoder. First, through variable length decoding (VLD), information for decoding of compressed video is obtained. The information from the 'VLD' block includes DCT coefficients, motion vectors (MVs), quantization

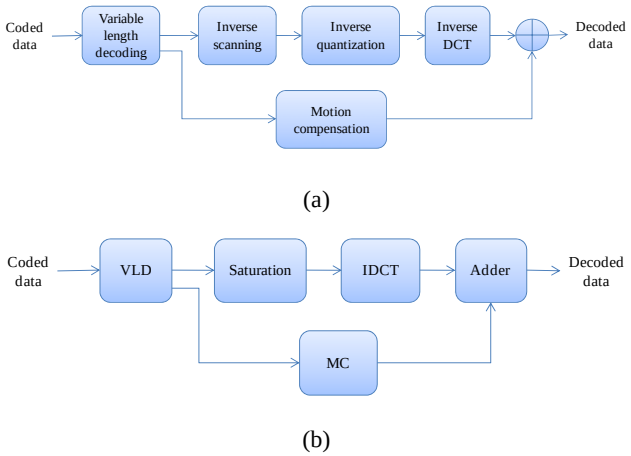

Fig. 1: (a) Block diagram of an MPEG-2 decoder, and (b) the modularized MPEG-2 decoder.

scale, types of pictures and macroblocks (MBs), mode information of decoding, and so on. DCT coefficients are transformed into residual data through inverse quantization (IQ) and inverse DCT (IDCT). If the data is intra-coded, the data after IDCT is just the raw data to be displayed. If predictive-coded or bidirectionally-predictive-coded, the data is the residual data to be added to block data that is motion-compensated from adjacent frames.

2.2 Modularization

We functionally modularize the decoder at the MB level into five blocks: 'VLD', 'Saturation', 'IDCT', 'MC', and 'Adder'.

'VLD' module includes not only VLD process, but also inverse scanning and IQ processes. 'Saturation' module includes saturation process to limit the coefficients for correct IDCT and mismatch control process. In 'Adder' module, the reference data is added to the residual data to obtain the final decoded image. Fig. 1(b) shows the block diagram of the modularized MPEG-2 decoder.

2.3 MPEG-2 MP@HL Decoder

MPEG-2 MP@HL is widely used as a standard for HDTV. Main profile supports a single layer stream, 4:2:0 color format, and three types of coding frames (I, P, and B). And high level constrains the resolution (1920 pixels/line and 1152 lines) and the frame rate (30 Hz). Table I shows the parameter limits of MPEG-2 MP@HL.

3. PARALLEL PROCESSING

3.1 Parallel Programming

Parallel programming is the design and implementation of parallel computer programs which can be used in parallel computing systems. For parallel programming, data/task/data-flow decomposition techniques are exploited.

For data decomposition, we divide the data into luminance components (Y) and chrominance components (C). Since the processes of Y and C are independent of each other, they can be well-separated. In order to separate the data, each task block except 'VLD' is spilt into two sub-blocks for processing of Y and C data.

We also take advantage of data-flow decomposition – pipelining. Pipelining is a parallelization technique in which multiple instructions are overlapped in execution. Efficiency of pipelining depends on how well the load of each stage is balanced. Five modules functionally divided cannot be processed with the same data simultaneously, but can be processed by pipelining because the data flows in the decoding process.

3.2 Performance

Performance of parallel processing is represented by Amdahl's law [13], which is expressed in terms of the speedup defined as

$$\text{speedup} = \frac{1}{S + P/n}, \tag{1}$$

where S and P represent serial and parallel portions, respectively ($S+P=1$), and n denotes the number of execution units, or the number of threads in our case. In (1), the speedup is greater than 1 if n is larger than one, and the speedup becomes higher when the parallel portion becomes larger.

In a real parallel system with the overhead, Amdahl's law can be represented as

$$\text{speedup} = \frac{1}{S + P/n + H(n)} \tag{2}$$

where $H(n)$ is the overhead of the parallel processing. If the overhead is big, the speedup becomes small. It can be even smaller than one.

If the system is parallelized by pipelining, (2) can be modified as

$$\text{speedup} = \frac{1}{\max(P_1, P_2, ..., P_n) + H(n)} \tag{3}$$

where P_i $(1 \le i \le n)$ represents the parallel portions corresponding to the i-th PE in the pipelining ($P_1+P_2+\ldots+P_n = 1$).

Table 1: Constraints of MPEG-2 MP@HL

No. of layers	Layer identifi-cation	Scalable mode	Maximum sample density (horizontal/verti-cal/frame)	Maximum sample rate (Hz)	Maximum total bit rate (bps)	Maximum total video buffering verifier buffer (bits)	Profile and level indication
1	0	Base	1920/1152/60	62,668,800	80,000,000	9,781,248	MP@HL

```
#pragma omp parallel sections
    #pragma omp section
        VLD
        put queue data
    #pragma omp section
        if (not first MB)
        {
            get queue data
            IQ
            IDCT
            put queue data
        }
    #pragma omp section
        if (neither first nor second MB)
        {
            get queue data
            MC
            Adder
        }
```

Fig 2: Pseudo-code of the multi-threaded code for Scenario 3 of MPEG-2 decoder.

3.3 Multi-Processor Environment

A multi-processing system uses two or more CPUs within a single computer system. In a multi-processing system, we can achieve a faster system by running multiple processes concurrently. We use the symmetric multiprocessing (SMP) system in which all the processors are identical and connected to the same shared memory.

DSP is widely used in a video codec since it is designed for real-time processing. The current technologies for DSP, such as lower design rules, fast-access cache, and a wider bus system, give a great performance. Some models use clock speed up to 1 GHz, perform eight operations per clock-cycle, or are capable of 8000 million instructions per second (MIPS) [14].

In this paper, we assume the multi-DSP environment. Although experiments are performed in a PC environment, we analyze the system as a multi-DSP system. It will give us more practical information because a multi-DSP system fits to the real-time decoding system. In our experiments, a multi-core, multi-processor system is used. The system is composed of two Dual-Core Intel® Xeon® processors – a total of four cores.

3.4 Implementation

For parallel programming, we utilize OpenMP which is an application programming interface (API) that supports shared memory multiprocessing programming in C/C++ and Fortran [15].

We use the MB-level parallelism as the thread granularity. At each MB, the coded data is decoded using parallel programming. Fig. 2 shows an example of the pseudo-code for Scenario 3, which will be described in Section 4.3. Each section in the code corresponds to each PE. Queue buffer is required for storing the data-flow. Since we use MB-level parallelism, a small space is needed only for MB decoding. We implement a circular queue whose length is equal to the number of stages of the pipeline.

4. SCENARIOS FOR PARALLEL MP@HL MPEG-2 DECODER

Using the parallel computing techniques mentioned in Section 3.1, we construct the parallel MP@HL MPEG-2 decoders. We propose five scenarios assuming that there are two or three DSPs for decoding processing. Fig. 3(a) shows the two-DSP system whereas Fig. 3(b) illustrates the three-DSP system. Table 2 lists the composition of each scenario for a multi-DSP MPEG-2 decoder.

4.1 Scenario 1

In this scenario, two DSPs are used. One DSP performs 'VLD' while the other performs the remaining operations – 'Saturation', 'IDCT', 'MC', and 'Adder'. This scenario is performed with a two-stage pipeline because the decoder consists of two PEs. This scenario takes advantage of data-flow decomposition with task partitioning.

4.2 Scenario 2

This scenario is almost the same as Scenario 1 except that 'Saturation' block is performed in the first DSP, not in the second.

4.3 Scenario 3

This scenario employs three DSPs. The first DSP performs 'VLD', and the second DSP performs 'Saturation' and 'IDCT' operations. The third DSP has 'MC' and 'Adder' functional blocks. This scenario is performed with a three-stage pipeline because the decoder consists of three PEs. It should be faster than Scenarios 1 and 2 due to the increased number of PEs while its cost also increases and it needs the larger bus bandwidth requiring faster clock speed.

4.4 Scenario 4

Scenario 4 is almost the same as Scenario 3 except that 'Saturation' block is in the first DSP, not in the second. The relation between Scenarios 3 and 4 is similar to that of

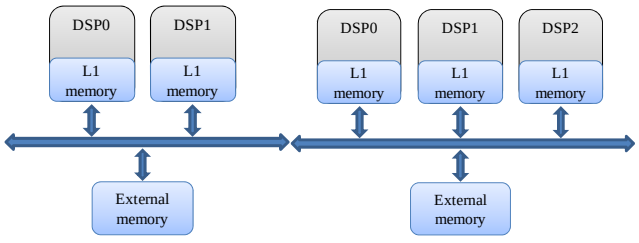

Fig. 3: MPEG-2 systems in a multi-DSP environment. (a) Two-DSP system, (b) three-DSP system.

Table 2: Composition of each scenario

	DSP0	DSP1	DSP2
Scenario 1	'VLD'	The rest	-
Scenario 2	'VLD', 'Saturation'	The rest	-
Scenario 3	'VLD'	'Saturation', 'IDCT'	'MC', 'Adder'
Scenario 4	'VLD', 'Saturation'	'IDCT'	'MC', 'Adder'
Scenario 5	'VLD'	The rest (Y)	The rest (C)

Scenarios 1 and 2.

4.5 Scenario 5

In the first DSP, 'VLD' is performed. The Y data and the C data are processed in the second and third DSPs, respectively. In this scenario, we take advantage of data decomposition method in addition to task and data-flow decompositions. Since decomposed data (Y and C) can be processed simultaneously in parallel, we can use a two-stage (not a three-stage) pipeline with three DSPs. So it needs only as large bus bandwidth as Scenarios 1 and 2 while the speed will increase. Unfortunately, the imbalance of data partitioning (Y:C=2:1) leads to some inefficiency.

5. PERFORMANCE ANALYSIS AND DISCUSSTIONS

In this section, we analyze the performance of the proposed decoder scenarios in view of the decoding speed, L1 memory size, and required bandwidth. For analysis and experiment, we use the encoded bitstream of the HD video 'Crossroad' (100 frames), which is encoded with MPEG-2 MP@HL.

5.1 Decoding Speed

Decoding speed is the most important performance measure in our cases, because our goal is to develop a fast MPEG-2 MP@HL decoder. First, we will take a look at the speed of the sequential decoder, and then estimate the speeds of parallel decoders based on the speed of the sequential one. Comparing the estimated speeds with the experimental results, we will analyze the simulation results.

5.1.1 Sequential decoder

In a sequential decoder, only a single thread is used to decode the bitstream. The observed decoding speed is shown in Table 3. The decoding speed is measured in cycles per MB.

5.1.2 Parallel decoder

Based on the speed of the sequential decoder, we calculate the speed of parallel decoders that are described in Section 4. It is assumed in the calculation that the sequential decoding is parallelized without any additional overhead such as threading overhead.

Table 3: Decoding cycles of a sequential decoder

Block		Decoding speed (cycles/MB)	Percentage (%)
'VLD'		35,323	32.8
'Saturation'	Y	8,099	7.5
	C	4,228	3.9
'IDCT'	Y	21,959	20.4
	C	10,723	10.0
'MC'	Y	12,003	11.2
	C	6,631	6.2
'Adder'	Y	5,600	5.2
	C	3,082	2.9
Total		107,648	100.0

Table 4: Decoding cycles of parallel decoders

Decoder type	Decoding speed (cycles/MB)	Speedup
Sequential decoder	107,648	1.00
Scenario 1	72,325	1.49
Scenario 2	59,998	1.79
Scenario 3	45,009	2.39
Scenario 4	47,650	2.26
Scenario 5	48,217	2.23

For example, in Scenario 1 'VLD' and the rest parts are performed by pipelining, in which the decoding time is calculated by max operation: max((35,323), (72,325)) = 72,325 cycles/MB. Scenario 3 has three task blocks: ('VLD'), ('Saturation', 'IDCT'), and ('MC', 'Adder'). Thus, the decoding speed is given by max((35,323), (8,099 + 4,228 + 21,959 + 10,723), (12,003 + 6,631 + 5,600 + 3,082)) = 45,009 cycles/MB. Table 4 lists the expected decoding time and the corresponding speedup of the proposed parallel decoders.

Using two DSPs, the improvement by factor of 1.79 (Scenario 2) can be expected while we can speed up by a factor of 2.39 (Scenario 3) with three DSPs. If the load distribution is even, the expected value will be the same as the number of DSPs employed.

5.1.3 Experimental results and analysis

As mentioned in Section 3.4, we implemented the five scenarios of MPEG-2 decoder using OpenMP. However, they are not performed in real multi-DSP environments. Thus, there exist other factors which are not needed in the multi-DSP environments.

The observed decoding speeds are shown in Table 5. As shown in Table 5, all the speedups are smaller than one, which means that parallel decoders are slower than the sequential one unlike those expected in the previous section. This is resulted from the fact that the experiments are performed in PC environments rather than in multi-DSP systems. The reason of the inconsistency can be summarized in three factors: fluctuation of the load, data queuing, and threading overhead.

First, fluctuation of the load degrades the performance. We divide the task blocks based on the averaged decoding time of each block. However, the load on each block changes from MB to MB according to the MB type. In order to

Table 5: Decoding speed from experiments

Decoder type	Decoding speed (cycles/MB)	Speedup
Sequential decoder	116,898	1.00
Scenario 1	159,234	0.73
Scenario 2	150,420	0.78
Scenario 3	131,266	0.89
Scenario 4	128,393	0.91
Scenario 5	137,929	0.85

Table 6: Decoding loads (cycles/MB) of each MB type

	Skip	Intra	Pred[1]	Pred[2]
'VLD'	6,800	58,076	35,041	38,503
	9.2%	52.2%	32.8%	29.1%
'Saturation'	10,113	10,197	10,246	10,263
	13.7%	9.2%	9.6%	7.7%
'IDCT'	33,649	34,788	34,253	34,202
	45.7%	31.2%	32.0%	25.8%
'MC'	14,079	123	18,038	40,055
	19.1%	0.1%	16.9%	30.2%
'Adder'	9,069	8,147	9,417	9,513
	12.3%	7.3%	8.8%	7.2%
Total	73,711	111,332	106,996	132,536
	100%	100%	100%	100%

[1] unidirectional prediction mode

[2] bidirectional prediction mode

check the fluctuation of the load, we measure the load of each module in view of the MB type. As in Table 6, the amount of time to perform 'VLD' and 'MC' depends on the MB mode, whereas 'Saturation', 'IDCT', and 'Adder' need the same amount of time. 'VLD' module takes the least time in Skip mode, and the most time in Intra mode. The speed of 'MC' block is dependent on the prediction mode.

Second, data-queuing for pipelining needs additional time to the time of decoding process. To preserve the data-flow, queue memory is required, which takes time to put data to and to get data from the queue. Table 7 shows the load from queuing, in which 'Put1' and 'Get1' operations carry the whole data of an MB, whereas 'Put2' just the coefficient data.

Third, there exists threading overhead in PC environments. These overheads are from creating, managing, and removing threads. To measure the threading overhead, we measure the time to be taken in the threading code (OpenMP), which is dependent on the number of threads as shown in Table 8. Speedups without threading overhead are shown in Table 9. Decoding speeds in Table 9 are calculated by simply subtracting threading loads from decoding speeds in Table 5. If there is no threading load, the speedup is greater than one in every scenario.

5.2 L1 Memory

Level 1 (L1) cache is on-chip memory that exists in the processor. Each DSP should have suitable L1 memory space enough to store the temporary data for decoding of each MB. If the size of L1 memory is too small, data transfer operations from external memory occur frequently, so the speed is reduced. Since we use an MB-level

Table 7: Queuing loads (cycles/MB)

Put1	Get1	Put2
4,016	3,835	1,181

Table 8: Threading loads (cycles/MB)

Scenarios 1/2	54,850
Scenarios 3/4/5	66,139

Table 9: Decoding speed without threading overhead

Decoder type	Decoding speed (cycles/MB)	Speedup
Sequential decoder	116,898	1.00
Scenario 1	104,384	1.12
Scenario 2	95,570	1.22
Scenario 3	65,127	1.79
Scenario 4	62,254	1.88
Scenario 5	71,790	1.63

Table 10: L1 memory size (bytes) of each scenario

	DSP0	DSP1	DSP2	Total
Scenario 1	1,163	2,780	-	3,943
Scenario 2	1,163	2,780	-	3,943
Scenario 3	1,163	1,536	2,012	4,711
Scenario 4	1,163	1,536	2,012	4,711
Scenario 5	1,163	1,960	888	4,011

parallelism, it is desirable that the L1 memory be as big as the data required for decoding an MB. In Table 10, the required L1 memory sizes of each DSP in the proposed scenarios are listed.

As shown in Table 10, the required cache size is smaller than 5 KB in all the scenarios. Since most of processors provide the cache memory bigger than 16 KB, all required sizes are small enough to use.

5.3 Bus Bandwidth

The proposed multi-DSP system is composed of multiple DSPs, external memory, and shared bus which connects each component. Bus bandwidth, which is defined by the amount of transferred data per second, is considered in two cases here.

In the first case, it is assumed that the only way for data transfer is via the shared bus, through which all the data are delivered. In the second case, it is assumed that there exist data paths between DSPs unlike the first case. Bus bandwidth is determined only by the amount of data transferred between DSP and external memory in this case. The bus bandwidths of each scenario in two cases are listed in Table 11. Also, bus clock speeds corresponding to each scenario are given in Table 12. The bus bandwidths are divided by bus width (32 bits) to give the bus clock speeds.

In the first case, Scenarios 1 and 2 require the lowest bus clock speed (142 MHz) and Scenarios 3 and 4 demands the highest bus clock speed (193 MHz). And if there exist additional paths between DSPs, lower bus clock speed (92 MHz) is required. The required bus clock speeds do not exceed 200 MHz, which means that the system is usable in

Table 11: Bus bandwidth (Mbps) of each scenario

	Case 1	Case 2
Scenario 1	4,557	2,943
Scenario 2	4,557	2,943
Scenario 3	6,171	2,943
Scenario 4	6,171	2,943
Scenario 5	4,666	2,943

Table 12: Bus clock speed (MHz) of each scenario

	Case 1	Case 2
Scenario 1	142	92.0
Scenario 2	142	92.0
Scenario 3	193	92.0
Scenario 4	193	92.0
Scenario 5	146	92.0

a view of bus bandwidth.

5.4 Choice of the Scenario

The choice of scenario should be made with consideration of the specifications of the system to be used. First, in aspect of the speed, the best scenario is Scenario 3 or 4. In other words, they divide the load the most evenly among five scenarios. Second, in terms of the memory, Scenarios 1 and 2 are better. They require the least amount of L1 cache memory. However, the memory required in every scenario is small enough that we can neglect it. Third, in respect to bus bandwidth, Scenarios 1 and 2 are the best while Scenarios 3 and 4 are the worst.
In addition, the cost can be considered. The cost of the given scenarios depends on the number of processors used. Therefore, the cost of Scenarios 3, 4, and 5 is higher than that of Scenarios 1 and 2.

6. CONCLUSIONS

In this paper, we compose five scenarios of MPEG-2 MP@HL decoder in multi-DSP environment and analyze the performance of each scenario in terms of decoding speed, cache memory size, and bus bandwidth. Although simulations in PC environment result in low speedups, our analysis shows the efficiency of the proposed system in multi-DSP environment. Parallel computing techniques in appropriate environment raise the performance of system. Future research will be on the similar works on the latest codec such as H.264.

7. ACKNOWLEDGEMENT

This work was supported in part by Samsung Advanced Institute of Technology (SAIT).

8. REFERENCES

[1] ISO/IEC 13818-2, Generic Coding of Moving Pictures and Associated Audio Information: Video, Nov. 1994.
[2] A. Cugnini and R. Shen, “MPEG-2 video decoder for the digital HDTV Grand Alliance system,” IEEE Trans. Consumer Electronics, vol. 41, no. 3, pp. 748−753, Aug. 1995.
[3] C. L. Lee, et al., “Implementation of digital HDTV video decoder by multiple multimedia video processors,” IEEE Trans. Consumer Electronics, vol. 42, no. 3, pp. 395−401, Aug. 1996.
[4] H. Yamauchi, et al., “Single chip video processor for digital HDTV,” IEEE Trans. Consumer Electronics, vol. 47, no. 3, pp. 394−404, Aug. 2001.
[5] N. Ling and N.-T. Wang, “A real-time video decoder for digital HDTV,” J. VLSI Signal Processing, vol. 33, no. 3, pp. 295−306, Mar. 2003.
[6] Y. Hu, A. Simpson, K. McAdoo, and J. Cush, “A high definition H.264/AVC hardware video decoder core for multimedia SoC’s,” in Proc. IEEE Int. Symposium Consumer Electronics, pp. 385−389, Reading, UK, Sep. 2004.
[7] T.-W. Chen, et al., “Architecture design of H.264/AVC decoder with hybrid task pipelining for high definition videos,” in Proc. IEEE Int. Symposium Circuits and Systems, vol. 3, pp. 2931−2934, Kobe, Japan, May 2005.
[8] T.-C. Chen, et al., “Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder,” IEEE Trans. Circuits and Systems for Video Technology, vol. 16, no. 6, pp. 673−688, June 2006.
[9] A. Bilas, J. Fritts, and J. P. Singh, “Real-time parallel MPEG-2 decoding in software,” in Proc. Parallel Processing Symposium, pp. 197−203, Geneva, Switzerland, Apr. 1997.
[10] E. B. van der Tol, E. G. T. Jaspers, and R. H. Gelderblom, “Mapping of H.264 decoding on a multiprocessor architecture,” in Proc. SPIE Conf. Image and Video Communications and Processing, vol. 5022, pp. 707−718, Santa Clara, CA, USA, Jan. 2003.
[11] Y.-K. Chen, X. Tian, S. Ge, and M. Girkar, “Towards efficient multi-level threading of H.264 encoder on Intel hyper-threading architectures,” in Proc. IEEE Parallel and Distributed Processing Symposium, pp. 63−72, Santa Fe, NM, USA, Apr. 2004.
[12] T. R. Jacobs, V. A. Chouliaras, and D. J. Mulvaney, “Thread-parallel MPEG-2, MPEG-4 and H.264 video encoders for SoC multi-processor architectures,” IEEE Trans. Consumer Electronics, vol. 52, no. 1, pp. 269−275, Feb. 2006.
[13] G. Amdahl, “Validity of the single processor approach to achieving large-scale computing capabilities,” in Proc. American Federation of Information Processing Societies Conf., vol. 30, pp. 483−485, Atlantic City, NJ, USA, Apr. 1967.
[14] C6000TM High Performance DSPs, http://www.ti.com.
[15] OpenMP Architecture Review Board, OpenMP Application Program Interface. Version 2.5, May 2005, http://www.openmp.org.