Multilevel Parallelization on the Cell/B.E. for a Motion JPEG 2000 Encoding Server

Hidemasa Muta
Tokyo Research Laboratory
IBM Japan
1623-14 Shimotsuruma
Yamato, Kanagawa
242-8502 Japan
hmuta@jp.ibm.com

Munehiro Doi
Global Engineering Solutions
IBM Japan
800 Ichimiyake Yasu-cho
Yasu, Shiga
520-2392 Japan
munepi@jp.ibm.com

Hiroki Nakano
Global Engineering Solutions
IBM Japan
800 Ichimiyake Yasu-cho
Yasu, Shiga
520-2392 Japan
hnakano@jp.ibm.com

Yumi Mori
Global Engineering Solutions
IBM Japan
1623-14 Shimotsuruma
Yamato, Kanagawa
242-8502 Japan
morip@jp.ibm.com

ABSTRACT

The Cell Broadband Engine (Cell/B.E.) is a novel multi-core microprocessor designed to provide high-performance processing capabilities for a wide range of applications. In this paper, we describe the world's first JPEG 2000 and Motion JPEG 2000 encoder on Cell/B.E. Novel parallelization techniques for a Motion JPEG 2000 encoder that unleash the performance of the Cell/B.E. are proposed. Our Motion JPEG 2000 encoder consists of multiple video frame encoding servers on a cluster system for high-level parallelization. Each video frame encoding server runs on a heterogeneous multi-core Cell/B.E. processor, and utilizes its 8 Synergistic Processor Elements (SPEs) for low-level parallelization of the time consuming parts of the JPEG 2000 encoding process, such as the wavelet transform, the bit modeling, and the arithmetic coding. The effectiveness of high-level parallelization by the cluster system is also described, not only for the parallel encoding, but also for scalable performance improvement for real-time encoding and future enhancements. We developed all of the code from scratch for effective multilevel parallelization. Our results show that the Cell/B.E. is extremely efficient for this workload compared with commercially available processors, and thus we conclude that the Cell/B.E. is quite suitable for encoding next generation large pixel formats, such as 4K/2K-Digital Cinema.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**]: Concurrent Programming – *Parallel programming.*

General Terms

Algorithms, Performance

Keywords

Parallelization, Motion JPEG 2000, Cell Broadband Engine

MM'07, September 23–28, 2007, Augsburg, Bavaria, Germany.

1. INTRODUCTION

Motion JPEG 2000 [1] is one of the ISO standard video formats that was published in Part 3 of the JPEG 2000 [2] ISO standard. It is expected to be a mastering video encoding format for digital cinema solutions because of its superior compression performance and support for easy editing of image content. A Motion-JPEG-2000-based digital cinema application profile [3] was also published as a standard digital cinema format. Each video frame in Motion JPEG 2000 is encoded as a JPEG 2000 data stream. JPEG 2000 can compress the image data at higher compression ratios without generating the blocky noise that appears in DCT-based JPEG compression. However, JPEG 2000's computation load is greater than the DCT-based JPEG. The digital cinema application profile defines 2 standard resolutions of 2048x1080 and 4096x2160. To encode one of the lower resolution (2048x1080) video frames, it takes more than 2 second per frame on a current-generation PC.

JPEG 2000 and Motion JPEG2000 encoding require large amounts of data processing and real-time processing, depending on the usage, such as medical image encoding or digital cinema encoding. We applied the computational power of the Cell/B.E. to the processing of Motion JPEG 2000. The Cell/B.E. allows effective solutions by multithread programming in ways that are not possible with single-core architectures.

In this paper we introduce our Motion JPEG 2000 encoder system which runs on a Cell/B.E.-based cluster system, and describe its parallelization techniques at various system levels and its encoding performance.

The reminder of this paper is organized as follows. In Section 2, we give an overview of our encoder system hardware and software. In Section 3, we show the performance profiling results of our single thread base code, and also introduce other existing implementations. In Section 4, we show our parallelization techniques at various system levels. In Section 5, we show the results of our parallelization and discuss them. We conclude in Section 6.

2. SYSTEM OVERVIEW

2.1 Introduction of Cell/B.E.

The Cell/B.E. is a multi-core chip consisting of two different types of processor elements, a PowerPC Processor Element (PPE) and multiple Synergistic Processor Elements (SPEs) [4]. The PPE is a general-purpose processor with a traditional virtual memory subsystem. It runs an operating system and handles the Cell/B.E.'s general workload and manages the special workloads for the SPEs. Each SPE consists of a Synergistic Processing Unit (SPU) and a 256-KB local store (LS). An SPE uses a special Single-Instruction Multiple-Data (SIMD) instruction set and relies on asynchronous Direct Memory Access (DMA) to move data between the main memory and the LS. The PPE and SPEs communicate with each other, with main storage, and with I/O devices through the Element Interconnect Bus (EIB). Figure 1 shows a Cell/B.E. block diagram. The Memory Interface Controller (MIC) and the Bus Interface Controller (BIC) provide the interfaces between the EIB and the main memory, and between the EIB and I/O, respectively. The Cell/B.E. also has an Internal Interrupt Controller (IIC). The programming scheme of the Cell/B.E. is different than used for single-core architecture. The programmer needs to take into account an appropriate design for application partitioning between the PPE and SPEs.

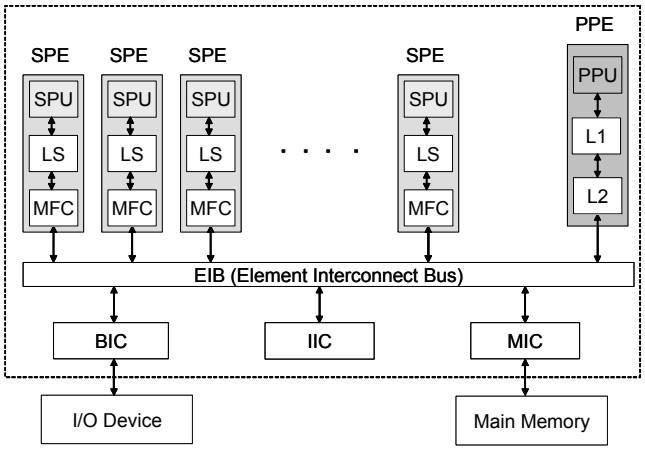

Figure 1. Cell Broadband Engine architecture

2.2 Cell/B.E.-Based Blade Server

The first generation Cell/B.E.-based system was released by IBM as a blade server in an IBM BladeCenter chassis. This blade server has 2 Cell/B.E. processors and 1 GB XDRAM memory on board. The 2 Cell/B.E. processors are connected as an SMP so that each PPE in the Cell/B.E. processor can access not only the entire main XDRAM memory but also the other Cell/B.E. processor's SPEs. Each Cell/B.E. processor has 8 SPEs in the chip, so this blade server enables a PPE to utilize up to 16 SPEs on the board. The clock speed of this first generation Cell/B.E. blade QS20 was 2.4 GHz [5].

The other features of the QS20 are:

- Peak performance of 410 GFlops (Billions of Floating point operations per second) in each Cell/B.E. blade
- A maximum of seven blades per chassis
- Two Gigabit Ethernet controllers provide connectivity to the BladeCenter chassis mid-plane and the BladeCenter Gigabit Ethernet switches

A PowerPC version of Linux (Fedora Core 5) runs on the PPEs of the Cell/B.E. processors on the blade server. To utilize the SPEs, we need to load the SPE code and invoke it from the PPE thread by using the special library LIBSPE.

2.3 Encoder Workload

Our Motion JPEG 2000 encoder is implemented as multiple software encoding servers which convert a PPM video frame to a JPEG-2000-encoded video frame. They run on multiple Cell/B.E. blade servers as a Linux application. They are called from an IA (Intel Architecture) blade server to encode each video frame of the Motion JPEG 2000 video stream. We present details in Section 4.

Before we show the final implementation, we begin by introducing our early implementation of a single thread JPEG 2000 encoder which works on the standard Linux environment. There was no parallelization in this implementation, but the encoding processes were optimized as well as possible.

The JPEG 2000 encoder is composed of 4 processes: (1) Preprocessing includes DC level shift and RCT (Reversible Component Transform) for lossless conversion, (2) Wavelet transform, (3) Quantization, which is necessary only for lossy conversion, and (4) EBCOT (Embedded Block Coding with Optimized Truncation) including bit modeling, arithmetic coding, and tag tree building, as shown in Figure 2.

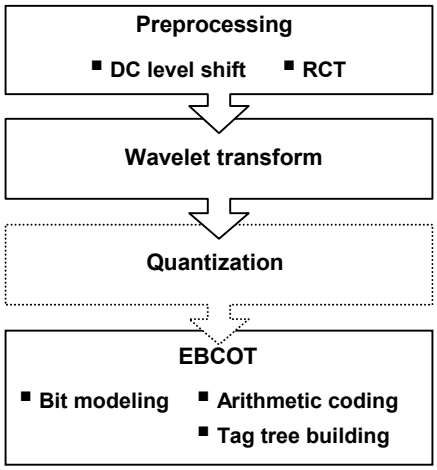

Figure 2. JPEG 2000 encoding processes

We verified the output data stream exactly matched the output from an existing implementation of a JPEG 2000 encoder, After that, we investigated which parts of the program were taking most of the execution time to determine which parts of the program to optimize for speed. For our work, the program was executed in the lossless conversion mode, and therefore the quantization process was not needed. The profiling results are shown in Section 3.

3. PROFILING

3.1 Single Thread Performance

We embedded the timer code into our single thread encoder to measure each process's performance. Table 1 shows the results of the performance measurement when we encoded one of the digital cinema standard 2,048x1,080 video frames with our single thread encoder. The encoder was compiled to use only the PPE on the Cell/B.E. blade server.

Table 1. Single thread performance on PPE

Preprocessing	Wavelet	EBCOT	Etc.
298 [ms]	1,075 [ms]	2,998 [ms]	33 [ms]

Internal timer total: 4,404 [ms]

System timer total: 7,608 [ms]

The file I/O access causes the difference between the internal timer result and the system timer result. The system timer includes the file I/O access time. Therefore we should focus on the internal timer results for the essential computation load of JPEG 2000 encoding. Our performance measurements showed the time consumption order of the processes: (1) EBCOT, (2) Wavelet, (3) Preprocessing, and (4) Etc. It also shows that most of the computation load is spent in the top 3 processes.

In addition, the time consumption ratio was investigated in detail by using a software profiling tool, "prof" that can list each function and how much of program execution time it used. Figure 3 shows the profile of the JPEG 2000 encoding logic, summarizing the results from the profiling results with the prof tool. The EBCOT process is split into "Bit modeling", "Arithmetic coding", and "Tagtree building, etc."

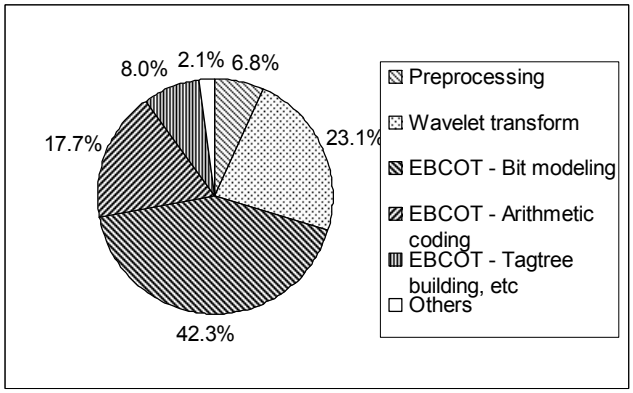

Figure 3. Profile of JPEG 2K encoding logic

Based on this performance analysis, we decided to implement the following high-load processes in the SPEs using the function offload model when we developed the JPEG 2000 code for Cell/B.E., starting from the code of our single thread encoder.

- Wavelet transform
- EBCOT – Bit modeling
- EBCOT – Arithmetic coding

3.2 Existing Implementations

We also measured the encoding performance of two other existing JPEG 2000 encoder implementations: JasPer 1.9 [6][7] and OpenJPEG 1.1.1 [8] to compare with our encoder's results on the same platform. Both of these implementations are distributed via websites as open source code for Linux platforms, so we were able to compile the code on the Cell/B.E. blade server without change. When we executed the binary code, the number of decomposition levels for the wavelet transform was set to 5, the same as in our encoder's logic.

- JasPer 1.9 on PPE

Parameters:

time ./jasper -f src.ppm -F result.jp2 -T jp2 -O numrlvls=5

-O mode=int --verbose

Internal timer total: 7,265 [ms]

System timer total: 8,288 [ms]

- OpenJPEG 1.1.1 on PPE

Parameters:

time ./image_to_j2k -i src.ppm -o result.jp2 -n 5

Internal timer total: 9,520 [ms]

System timer total: 15,536 [ms]

We assume that system timer count includes file I/O. But even though we compare the system timer value of our encoder (7608 [ms]) to these existing implementation's internal timer value (7265 [ms] and 9520 [ms]), the results show that our encoder's single thread performance is almost equal or better than the other existing implementations.

4. PARALLELIZATION

4.1 Parallelization within the Processes

4.1.1 DC level shift and RCT

The first step of encoding is a DC level shift and transformation to the YUV color space. Our reversible encoder uses a Reversible multiple Component Transformation (RCT). Because the DC level shift and RCT are very simple operations, we implemented it to run on the PPE to avoid the overhead of offloading it to the SPEs. We developed all of the code from scratch based on the JPEG 2000 specification, and it was optimized by using the PPE's Vector/Multimedia Extension (VMX) instruction set. VMX is a Single-Instruction Multiple-Data (SIMD) instruction set using 128-bit registers. Each pixel is expressed as a 16-bit integer so that one VMX instruction can handle 8 pixels per cycle in most cases. In some case we used two 32-bit instructions for accuracy.

4.1.2 Wavelet transform

The next step is the Reversible Forward Wavelet Transform (RFWT). This separates the data transformed from a source image with the DC level shift and RCT into a sub-band scaling coefficients and wavelet coefficients.

Because the Cell/B.E. has 8 SPEs, we implemented the RFWT based on a tiling strategy to maximize the utilization of the SPEs. The SPEs process tiles which are cut from a source data frame. Actually, in the JPEG 2000 specification, a tiling option is specified, but our goal is different from the regular options in the specification. The goal of the regular tiling option is to reduce processing granularity. In other words, the regular option allows the image to be considered as a set of subimages of the source image. This approach is good for memory consumption but bad for the quality of the boundaries between tiles. Our tiling approach is equivalent to processing the whole area as one tile. However our tiling reduces memory consumption and supports parallel processing by the SPEs without any quality problems. In this paper, we call a tile used by our approach a "pseudo-tile" to distinguish it from a tile used by the regular option.

The size of every pseudo-tile is 128 pixels by 128 pixels. This size was chosen for two reasons. One reason was to maximize the DMA performance using multiples of 128-byte transfers. We used 2 bytes per pixel so that in every row of a tile, each row has 128 pixels, which will naturally be a multiple of 128 bytes. The second reason was the capacity of the SPE's Local Store (LS). The LS is only 256 KB and it must contain both code and data, including any working area. Since we used a double buffering technique (described below), two tiles at a time must be stored in the LS.

The wavelet transformation is a kind of convolving operation, so each tile has to know the data in the neighborhood of each border. For this reason, the surrounding 8 pixels for each tile are overlapped with the neighboring tiles. The net size of the transformed area is 112 pixels by 112 pixels. This overlap size comes from the SIMD register size and the DMA boundary constraints. All of the SPE's registers are 16 bytes, and the DMA transfer address must be a multiple of 16 bytes. The 8 pixel-long extensions in our implementation are matched with these values. For tiles at the edge of the image, we replicate 8 pixels into an extension buffer in the area surrounding the source image (Figure 4(a)). Also, there may be incomplete tiles on the right and bottom edges of the source image. For these partial tiles, we add extended pixels to treat the tiles as being 112 pixels by 112 pixels (Figure 4(b)). Using these refinements, we can treat every tile as being the same size to simplify the implementation on the SPEs without worrying about the edges, the pixel replications, or the incomplete tiles.

Each SPE performs an X-axis transformation and a Y-axis transformation to separate the 4 small tiles containing the sub-band coefficients of LL, HL, LH, and HH (Figure 4(c)). After this transformation, the small tiles are transferred using DMA to the proper position for the next level image stored in main memory (Figure 4(d)).

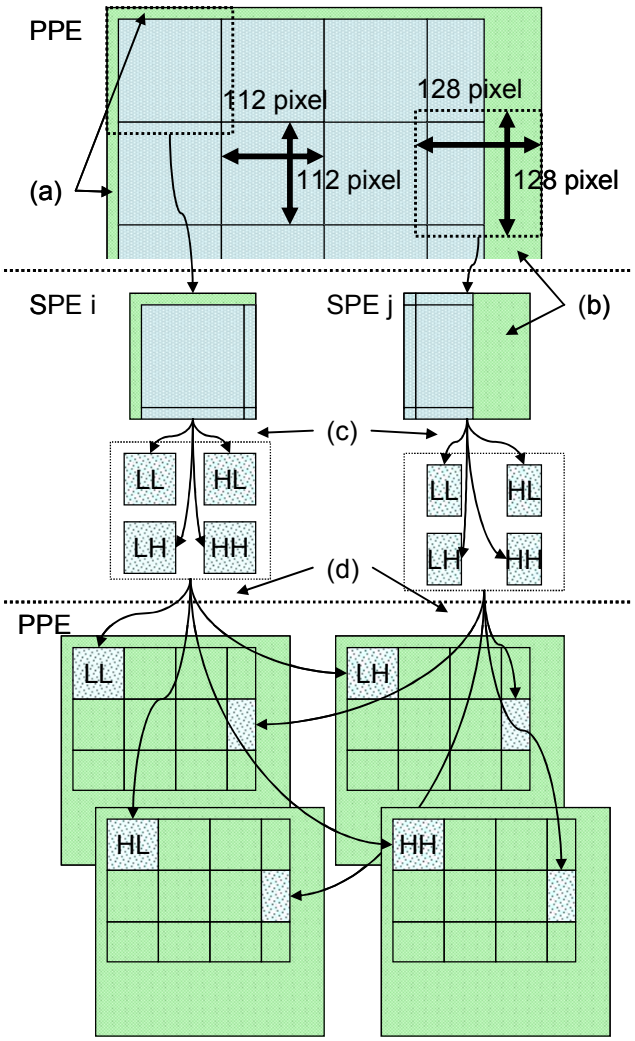

Figure 4. Parallelization of the wavelet transform

We applied 3 optimization techniques to the SPE code. First is the use of SIMD operations. As with the DC level shift and RCT, 8 pixels are processed at a time in the wavelet transform. The second technique is loop unrolling. Loop unrolling is a technique to expand the code in a loop structure instead of using a small for/while instruction. This can help the compiler check dependencies in the code so that the generated code is more

efficient because of better pipelining and better superscalar optimization. We unrolled 8 cycles of each x-axis loop and 4 cycles of each y-axis loop. As a result of SIMDization and loop-unrolling, the clock cycles to perform the 1D transformation for a tile were reduced to less than 2% of the original code (Figure 5). The third technique was double buffering. We use two buffers in the LS to perform DMA and calculations simultaneously. One of them does a DMA transfer while the other is being used for calculations (Figure 6). To support double buffering, one SPE processes a maximum of 8 tiles as 1 job unit.

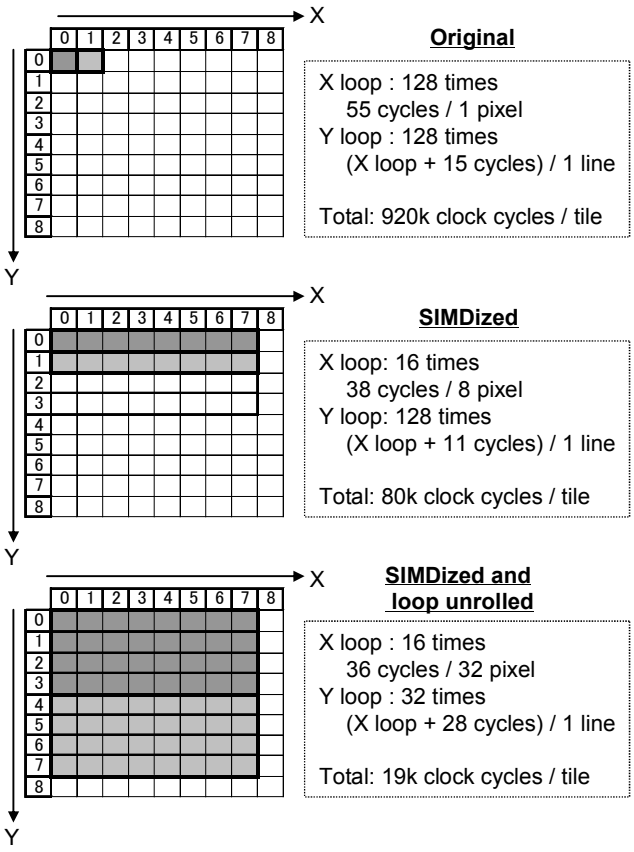

Figure 5. Optimization Results for 1D-RFWT on SPE

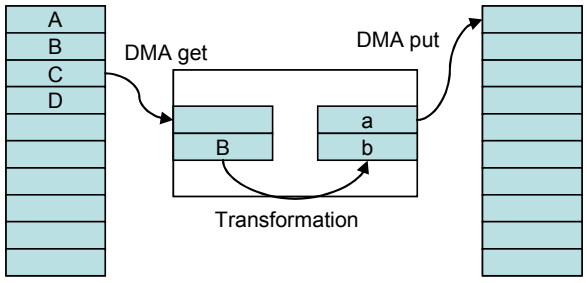

Figure 6. Double buffering

4.1.3 EBCOT

EBCOT (Embedded Block Coding with Optimized Truncation) involves bit modeling, arithmetic coding, and tag tree building. Both the bit modeling and the arithmetic coding are time consuming processes, as shown by the profiling results in Figure 3. The results of the wavelet transform process are the sub-band parameter arrays of LL, HL, LH, and HH. EBCOT divides these sub-band parameter arrays into small code blocks of 32x32 pixels in CPRL (Component-Position-Resolution Level-layer) order. Then it does bit modeling and arithmetic coding for each code block. When all of the code blocks for one of the four sub-bands have been completely processed, a tag tree is built from the compressed sizes of each code block. Thus, the tag tree building cannot be started before the bit modeling and the arithmetic coding are completed for all of those code blocks.

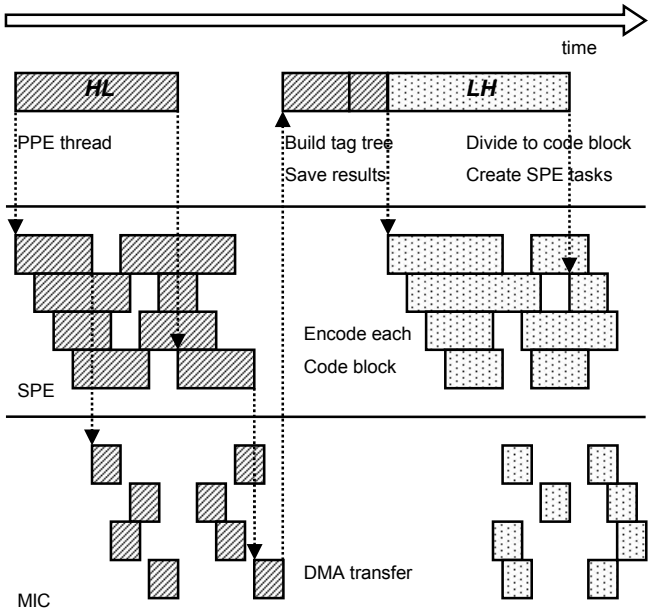

Figure 7. Single PPE thread parallelization of EBCOT

Figure 7 shows our early parallelization of EBCOT. In this implementation, we processed the bit modeling and arithmetic coding in parallel for each code block by using multiple SPEs. The sub-band parameter array was divided into code blocks by the PPE thread. Then the PPE thread assigned multiple SPEs to do the bit modeling and arithmetic coding for each code block. When the SPE completed the bit modeling and arithmetic coding, it copied the output data stream to the PPE's main memory with a DMA transfer. The SPE then waited until the DMA transfer was completed before changing its status to 'idle' so that PPE could assign it another code block for bit modeling and arithmetic coding. When all of the code blocks for a sub-band parameter array were completed in the SPEs, the PPE began to build the tag tree. After completing the tag tree, the PPE divided the next sub-band parameter array into code blocks.

In that implementation, we found the following areas that could be improved: 1) The SPEs did no work while the PPE was building a tag tree. 2) Each SPE waited until each DMA transfer was completed.

To improve the utilization of the SPEs, we used another PPE thread (a PPE consumer thread) to build the tag tree and to save the encoded results from the SPEs. This implementation allowed the PPE factory thread to assign an SPE for the next sub-band parameter array without waiting for the completion of tag tree building. We also added double output stream buffering for the SPE local memory, so that the SPE would work on the next code block without waiting for the completion of the DMA transfer. Figure 8 shows this improved implementation.

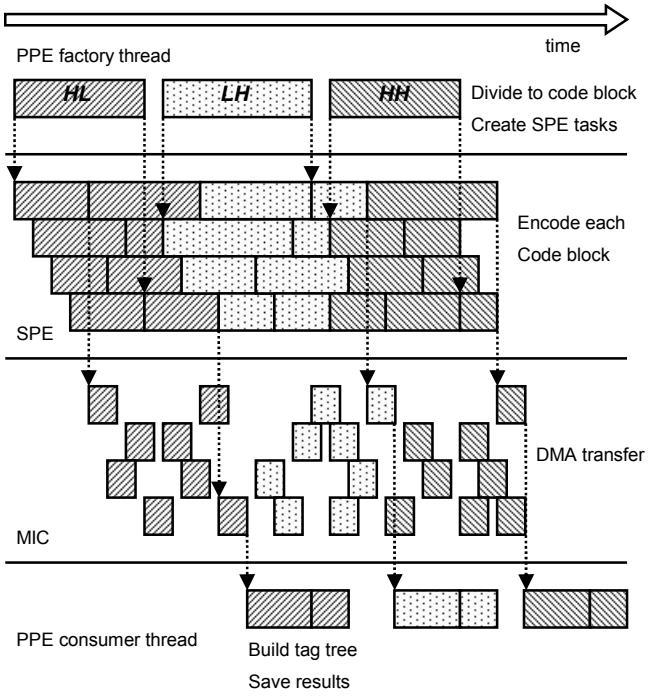

Figure 8. Double PPE threads parallelization of EBCOT

In this implementation, the processing times for tag tree building and DMA transfer are completely hidden. The utilization of the SPEs was increased to over 90%.

To implement this design, we arranged the code block information chain structure as shown in Figure 9. This chain is allocated by the PPE factory thread to assign code blocks to SPEs, and then the PPE consumer thread takes the results in the same order as the PPE factory thread assigned them, which is the same as the CPRL progression order. The processing time for the bit modeling and the arithmetic coding for each block depend on the complexity of the data, so the first SPE does not always complete its processing first. However, even though the tasks on each SPE are completed in random order, the PPE consumer thread can accept the results in order by following the chain and checking if the result was completely transferred from the SPE. After the PPE consumer thread has received all of the results, it frees an element of the chain. When it reaches the last element of the chain, all of the EBCOT processing has been completed.

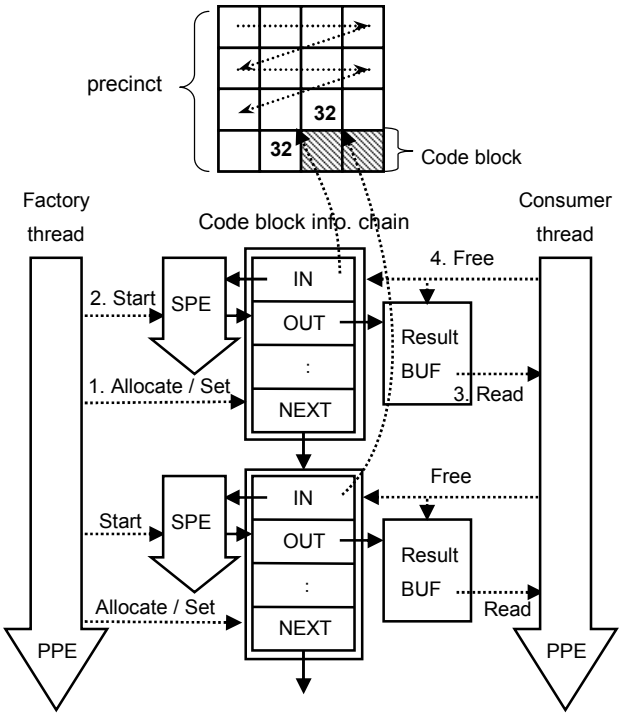

Figure 9. Code block information chain

Figure 10 shows the results of the performance measurements of these two implementations on the Cell/B.E. blade when they process a sub-band parameter array for 2,048x1,080 pixel images. For the 2-PPE-thread implementation, we tested with the double buffering enabled and disabled to measure the difference.

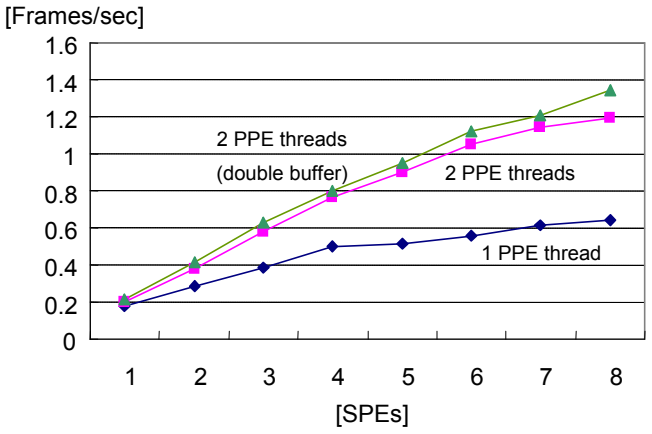

Figure 10. Performance improvement from 2-thread implementation

As shown in this figure, the 2-PPE-thread implementation is almost 2 times faster then the 1-PPE-thread implementation. The double buffer is effective when the number of SPEs is increased.

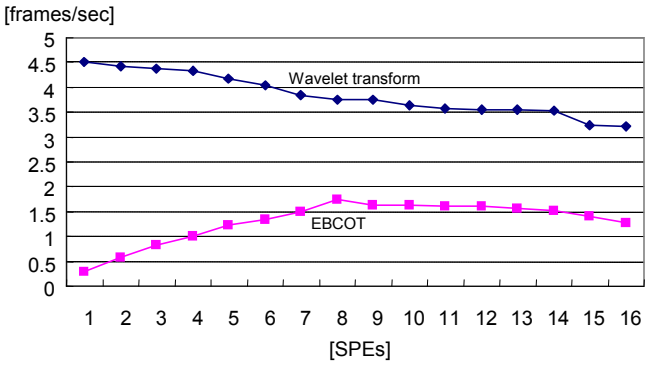

Figure 11. Performance of wavelet and EBCOT by 1-16 SPEs

4.2 Parallelization of the Processes

The key to performance improvements on the Cell/B.E. processor is the utilization of the SPEs. To maximize their utilization, we should not leave SPEs idle or use them inefficiently.

Figure 11 shows the performance of the wavelet transform and EBCOT on a Cell/B.E. blade when we allocated 1 to 16 SPEs for each process.

In the wavelet transform process, we found that using more SPEs does not improve the performance because of the synchronization issue as shown in Figure 12. Wavelet transformation needs synchronization at every transformation of each level to generate replicated pixels in periferal area. Additionally the number of tiles is getting smaller as progressing transformation because the transformed image (for example L1) is a half size of the source image (L0). After all, we have achieved 5.5 times faster performance on each SPE compared to the original PPE version.

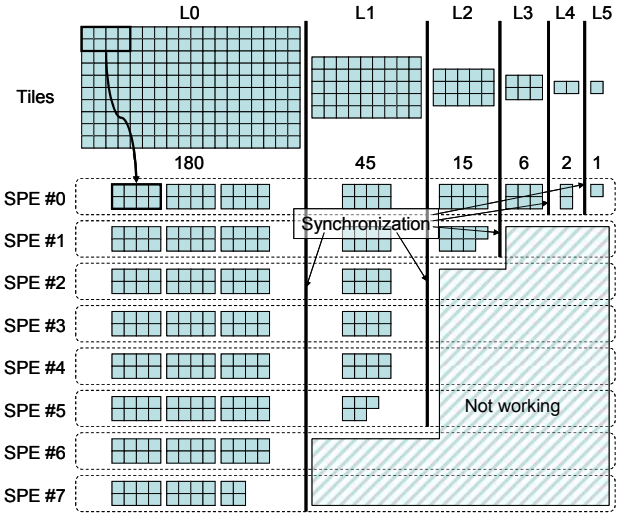

Figure 12. Synchronization issue of wavelet transformation

In the EBCOT process, the peak performance was recorded when we allocated 8 SPEs to it. This means the parallelization is effective as long as it takes place within one Cell/B.E. Processor.

When we use one Cell/B.E. processor for the encoding processes, we should consider how to better utilize the SPEs for the wavelet transform. We tried encoding 2 different video frames as 2 encoding threads, as shown in Figure 13. This implementation allows the wavelet transform and the EBCOT to run in parallel as different thread with semaphore synchronization. We allocated 1 SPE for the wavelet and 7 SPEs for the EBCOT so that the wavelet and the EBCOT could work in parallel on the same Cell/B.E. processor. While each encoding thread processes EBCOT, they create grandchild threads as a factory thread and a consumer thread.

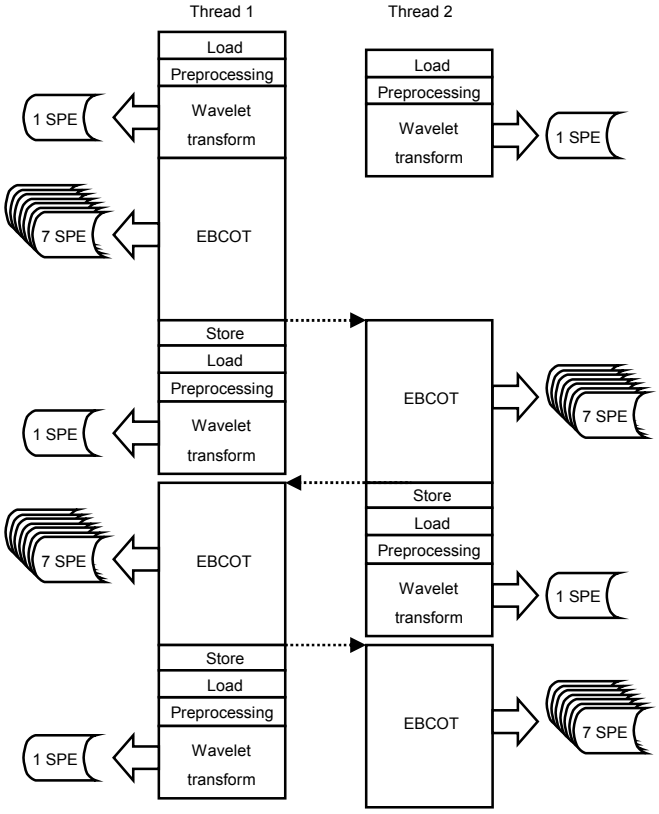

Figure 13. Parallelization of 2 encoding threads

4.3 Parallelization on the Blade

As described in Subsection 2.2, the Cell/B.E.-based blade server has 2 Cell/B.E. processors on board. To utilize these resources we have 2 options: execute 1 process on the 2 Cell/B.E. processors and utilize 16 SPEs for that process, or divide these Cell/B.E. processors as separate resources so that 2 different processes which utilize 8SPEs each can be executed. processors with 8 SPEs. As Figure 11 shows, the two processes that consume the most time for JPEG 2000 do not perform well even when we allow these processes to run on more than 8 SPEs. Therefore the two-process implementation is better for our encoder. This implementation is also effective in reducing congestion of the EIB (the internal bus of the Cell/B.E. processor). When we divide the Cell/B.E. processors using NUMA control with the following parameters, the EIB is not shared between the Cell/B.E. processors and the PPE utilizes only the 8 SPEs inside the same Cell/B.E. processor.

```
numactl –cpubind=0 –preferred=0 ./encoder –port 2000
numactl –cpubind=1 –preferred=1 ./encoder –port 2001
```

The NUMA control also divides the main XDRAM memory for each PPE so that each of them can access its own memory faster. The last parameter –port is the TCP/IP port used to accept JPEG 2000 encoding requests from the other process. These 2 processes work asynchronously on the blade running as the video frame encoding server. Each video frame encoding server is called from the client process on the other blade as described in Subsection 4.4.

4.4 Parallelization by Cluster System

Figure 14 shows our entire encoder system. It consists of multiple Cell/B.E. blade servers on which the JPEG 2000 frame-encoding server application runs, and an IA (Intel Architecture) blade server that runs the client process. The client process is the entry point for the PPM data stream coming from the video camera or from other storage. The PPM data frames are stored into the large cache memory on the IA blade, which goes up to 64 GB. The client process allocates the idle state video frame encoding servers on the Cell/B.E. blade servers to encode one of the video frames for the JPEG 2000 data stream. When it finishes encoding the video frame on the encoding server, the JPEG 2000 data stream is sent back to the IA blade, and stored into the cache. Then the client process combines these JPEG 2000 encoded data streams into the Motion JPEG 2000 data stream with the header and metadata from the encoded frames. The reason why the IA blade stores the encoded data stream into the cache memory again is that the frame encoding time depends on the complexity of the image of each frames, so to link the encoded frames in order, it needs to wait until the oldest frame is ready. The cache is also effective in masking the delays of the network and storage. Each Cell/B.E. blade and the IA blade are connected by a Giga-bit Ethernet, so the data transfer between these blades is much faster than the encoding time.

This cluster system also supports system enhancements. To improve the performance of this system, we can just add Cell/B.E. blades to the cluster system, and run the video frame encoding server on them.

5. RESULTS

5.1 Encoder Performance

As described in Section 4, our Motion JPEG 2000 encoding system was improved by parallelization techniques at various levels. To confirm the performance improvements of each parallelization technique, we temporally disabled some of the encoder code for optimizations. Then we ran those versions on the same hardware consisting of 1 Cell/B.E. blade server and 1 IA blade server, with the same 24 frames video content.

The first version is a single thread video frame encoder which disables all of the SPE code and SIMD code for preprocessing. It uses only the PPE on the Cell/B.E.-based blade server and is called from the client process on the IA blade server. Table 2 shows the total time for encoding 24 frames using this system. This result is the base performance to evaluate the performance improvements of the other versions.

The second version is a multithread video frame encoder which enables parallelization by using 16 SPEs on a Cell/B.E. blade. The SIMD code for preprocessing is also enabled. As in the first version, the client process in the IA blade server calls this process on the Cell/B.E. blade. Table 3 shows the results for encoding 24 frames with this version. The encoding performance is nearly 3 times faster compared to the single thread system. In particular, the processing time of EBCOT is more than 4 times faster.

The third version is the multithread video frame encoder using 8 SPEs on each Cell/B.E. processor. As described in Subsection 4.3, the 2 processes of this encoder work in parallel by using NUMA control on the Cell/B.E. blade. These 2 video frame encoding processes are called in parallel from the client process on the IA blade server. Table 4 shows the results for encoding 24 frames with this version. The encoding performance is more than double the 16-SPE implementation because of the utilization improvements of the SPEs for the wavelet transform and EBCOT.

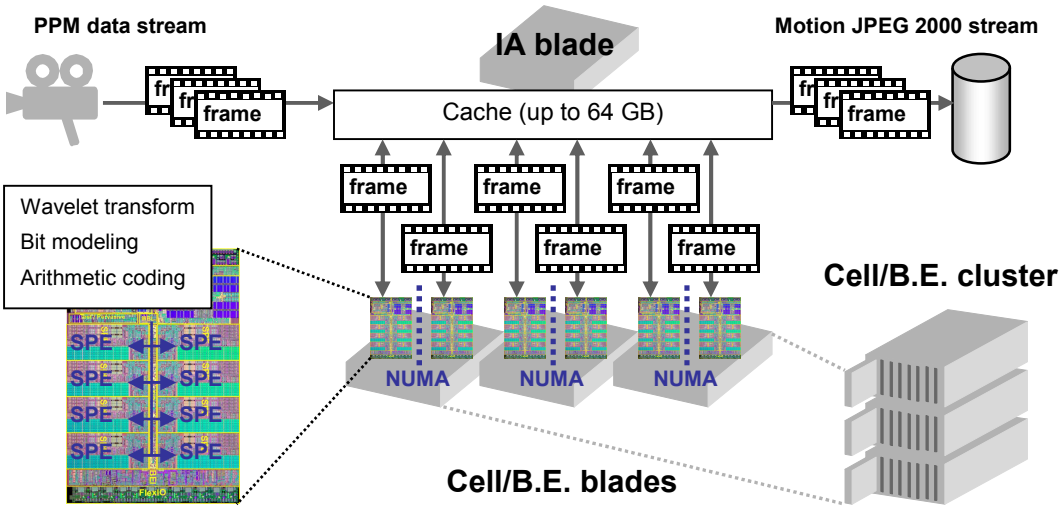

Figure 14. Cluster system

Table 2. 24 frames encoded by 1 PPE thread

Preprocessing	Wavelet	EBCOT	Etc.
7,681 [ms]	28,522 [ms]	70,893 [ms]	9,117 [ms]

Total: 116,213 [ms] (0.21 [frames/sec])

Table 3. 24 frames encoded by 1 PPE and 16 SPE threads

Preprocessing	Wavelet	EBCOT	Etc.
2,195 [ms]	9,331 [ms]	17,504 [ms]	10,051 [ms]

Total: 39,081 [ms] (0.61 [frames/sec])

Table 4. 24 frames encoded by dual 1 PPE and 8 SPE threads

Preprocessing	Wavelet	EBCOT	Etc.
1,010 [ms]	3,531 [ms]	9,132 [ms]	5,239 [ms]

Total: 18,912 [ms] (1.27 [frames/sec])

5.2 Comparison with Other Implementations

We also measured the performance of the existing JPEG 2000 encoders JasPer 1.9 and OpenJPEG 1.1.1 on the IA PC when they encode same 24 video frames we used for Cell/B.E. encoder. The total encoding time was calculated from the application timer results for each video frame.

- JasPer 1.9 on IA PC

Total encoding time for 24 frames:

48,744 [ms] (0.49 [frames/sec])

CPU (Memory): Pentium 4 - 3.4 GHz (1 G bytes)

OS (Compiler): Windows XP SP2 (gcc on Cygwin)

- OpenJPEG 1.1.1 on IA PC

Total encoding time for 24 frames:

75,408 [ms] (0.32 [frames/sec])

CPU (Memory): Pentium 4 - 3.4 GHz (1 G bytes)

OS (Compiler): Windows XP SP2 (Visual C++ 6.0)

They only encoded 24 video frames to JPEG 2000 data stream, so the results do not include the time for Motion JPEG 2000 format building. Also, the file I/O overhead should be considered depending on whether or not their application timer includes it.

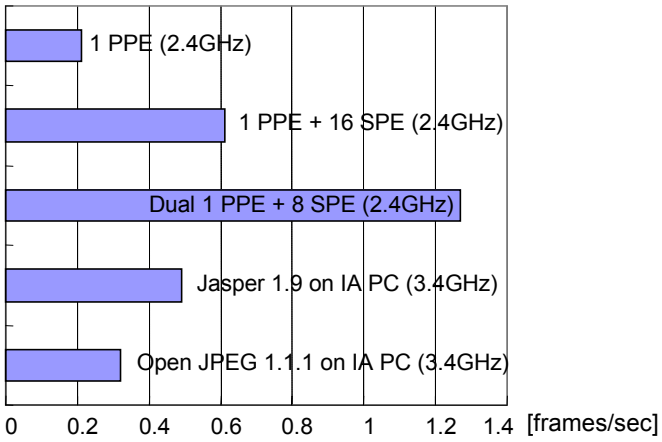

Figure 15. Encoding performance

Figure 15 shows the frames per second performance of these encoders. The dual 1 PPE and 8 SPE encoder is more than 6 times faster than the base 1 PPE code on the same hardware. It is also more than 2 timers faster than the existing encoders on an IA PC. The performance can be further improved when we use the latest QS20 Cell/B.E. blade having 3.2GHz clock frequency.

5.3 Cluster System Performance

To this point, we evaluated the performance and utilization for a one-blade Cell/B.E. blade server. Next, we evaluated the scalability of our encoding server system by increasing the number of Cell/B.E. blades.

Tables 5 and 6 show the results for encoding 24 frames and 240 frames with multiple Cell/B.E. blade servers. The dual-process multithread video frame encoders ran on each of the Cell/B.E. blades. This version was 8% slower than the comparable results included in Table 4 because of the insufficient optimization of EBCOT parallelization.

Table 5. 24 frames encoded by multiple Cell/B.E. blades

# of blades	1 blade	2 blades	3 blades
Total time	20,399 [ms]	11,685 [ms]	7,718 [ms]

Table 6. 240 frames encoded by multiple Cell/B.E. blades

# of blades	1 blade	2 blades	3 blades
Total time	219,359 [ms]	107,070 [ms]	65,945 [ms]

In this performance evaluation, we focused on the scalability of the results. As figure 16 shows, the performance was improved linearly when we increased the number of Cell/B.E. blades. This means there are no bottlenecks in the cluster system as long as the data transfers between the client process and the video frame encoding servers are not saturated. The frame rate performance of the 24 frames encoding and the 240 frames encoding is almost the same. This shows the stability of this multiple-blade encoding system for the real content.

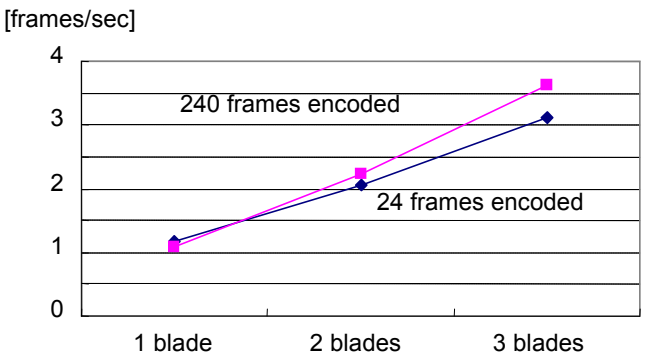

Figure 16. Multiple blades encoding results

6. CONCLUSION AND FUTURE WORK

In this paper, we described the world's first Cell/B.E.-based JPEG 2000 and Motion JPEG 2000 encoder system. We gave an overview of our encoder system hardware and software on Cell/B.E. blade servers, showed the performance profiling results of our single thread code on PPE, introduced several implementations with different number of SPEs, and showed our parallelization techniques at various system levels and their results.

First, the single thread task profile on PPE was analyzed and we found that wavelet transform, bit modeling, and arithmetic coding are the time consuming tasks. Based on this, we developed various optimization techniques for Cell/B.E. The techniques which we used are:

- SIMDization of the wavelet transform task on a single SPE.
- Parallelization of the EBCOT task using multiple SPEs.
- Resource utilization optimization between the wavelet and EBCOT tasks.
- High-level parallelization by configuring the Cell/B.E. blade cluster system.

SIMDization, double buffering, and loop unrolling on a single SPE were efficient in the wavelet transform. In the EBCOT task, we introduced a PPE factory thread to assign SPEs for code block encoding, and a PPE consumer thread to build tag trees and save the encoded results coming from the SPEs. A double buffering technique was also effective for the EBCOT task on the SPEs. By using this implementation, the processing times of tag tree building and DMA transfers were completely hidden, and the utilization ratio of SPEs was boosted above 90%. In the EBCOT tasks, the peak performance was recorded when we allocated 8 SPEs. Therefore we allocated 1 SPE for the Wavelet and 7 SPEs for the EBCOT so that the Wavelet and the EBCOT could work in parallel on one Cell/B.E. We also constructed a cluster system which consisted of Cell/B.E. blades and an IA blade. The IA blade dispatches the frame encoding tasks to idle Cell/B.E. blades. We found that the high-level parallelization on the cluster system was effective, not only for the parallel encoding, but also for scalable performance improvements for real-time encoding and future enhancements. We developed all of the code from scratch for effective multilevel parallelization. Our results show that the Cell/B.E. is extremely efficient for this processing compared with commercially available processors, and thus, we conclude that the Cell/B.E. is quite suitable for encoding next generation large pixel formats, such as 4K/2K-Digital Cinema. Our future work will include further optimizations of EBCOT task on the SPEs, implementation of the 4K digital cinema format, and nonreversible encoding mode support.

7. ACKNOWLEDGMENTS

Our thanks to Junji Maeda of IBM Business Consulting Service for prototyping the early implementation of the single thread JPEG 2000 encoder logic.

We also thank Hiroki Nishiyama of IBM Japan for arranging for the Cell/B.E. blade server and Cell/B.E. software development environment.

8. REFERENCES

[1] ISO/IEC 15444-3, "Information technology - JPEG 2000 image coding system - Part3: Motion JPEG 2000"

[2] ISO/IEC 15444-1, "Information technology - JPEG 2000 image coding system - Part1: Core coding system"

[3] ISO/IEC 15444-1, "Information technology - JPEG 2000 image coding system - Amendment 1: Profiles for digital cinema applications"

[4] M. Gschwind et al, "Synergistic Processing in Cell's Multicore Architecture", *IEEE Micro March 2006*, 2006

[5] http://www-03.ibm.com/technology/splash/qs20

[6] Michael D. Adams and Faouzi Kossentini, "JasPer: a software-based JPEG-2000 codec implementation", *IEEE International Conference on Image Processing 2000*, Vol. 2, pp.53-56, 2000

[7] Michael D. Adams and Rabab K. Ward, "JasPer: a portable flexible open-source software tool kit for image coding/processing", *IEEE International Conference on Acoustics, Speech, and Signal Processing 2004*, Vol. 5, pp.17-21, 2004

[8] http://www.openjpeg.org

9. COPYRIGHTS