Optimizing JPEG2000 Still Image Encoding on the Cell Broadband Engine

Seunghwa Kang David A. Bader
Georgia Institute of Technology, Atlanta, GA 30332
s.kang@gatech.edu bader@cc.gatech.edu

Abstract

JPEG2000 is the latest still image coding standard from the JPEG committee, which adopts new algorithms such as Embedded Block Coding with Optimized Truncation (EBCOT) and Discrete Wavelet Transform (DWT). These algorithms enable superior coding performance over JPEG and support various new features at the cost of the increased computational complexity. The Sony-Toshiba-IBM Cell Broadband Engine (or the Cell/B.E.) is a heterogeneous multicore architecture with SIMD accelerators. In this work, we optimize the computationally intensive algorithmic kernels of JPEG2000 for the Cell/B.E. and also introduce a novel data decomposition scheme to achieve high performance with low programming complexity. We compare the Cell/B.E.'s performance to the performance of the Intel Pentium IV 3.2 GHz processor. The Cell/B.E. demonstrates 3.2 times higher performance for lossless encoding and 2.7 times higher performance for lossy encoding. For the DWT, the Cell/B.E. outperforms the Pentium IV processor by 9.1 times for the lossless case and 15 times for the lossy case. We also provide the experimental results on one IBM QS20 blade with two Cell/B.E. chips and the performance comparison with the existing JPEG2000 encoder for the Cell/B.E.

1 Introduction

JPEG2000 [4] is the latest still image coding standard issued by the JPEG committee, which supports both lossless and lossy compression with superior image quality in a low bit rate and additional new features. This standard adopts Embedded Block Coding with Optimized Truncation (EBCOT) [13] and Discrete Wavelet Transform (DWT) [2, 14] as key algorithms. The EBCOT algorithm consists of three steps: bit modeling, arithmetic coding, and tag tree building. JPEG2000 executes the EBCOT algorithm in two tiers, Tier-1 and Tier-2. Bit modeling and arithmetic coding are performed in Tier-1, whereas tag tree building is performed in Tier-2. Prior analysis of JPEG2000 execution time [1, 8, 9] shows that Tier-1 coding in the EBCOT and the DWT are the most computationally expensive algorithmic kernels.

The Cell Broadband Engine (or the Cell/B.E.) has unique architectural features with a simple core design and an alternative memory subsystem. The Cell/B.E. chip consists of two types of cores, one PPE and eight SPEs. The PPE is a power efficient version of PowerPC architecture, and the SPE is a SIMD accelerator. The SPE lacks dynamic branch prediction and runtime out-of-order execution support. It has a 256 KB local memory called Local Store. Data transfers among main memory and the Local Stores require explicit DMA instructions. This necessitates novel programming paradigms to achieve high performance.

Previously, Muta et al. optimize Motion JPEG2000 encoder on the Cell/B.E. [10]. Motion JPEG2000 encoding lacks inter-frame compression and is nearly identical to JPEG2000 still image encoding, and the authors optimize the DWT and EBCOT algorithms. However, their DWT implementation does not scale beyond a single SPE despite having high single SPE performance. Their EBCOT implementation shows better scalability but does not scale above a single Cell/B.E. processor. This suggests that in order to achieve high performance, we need to take different approaches.

Our Cell/B.E. JPEG2000 library adopts Jasper [1], a still image transcoder, as a baseline program. Jasper is previously parallelized by Meerwald et al. [9] using OpenMP. However, the authors parallelize Tier-1 coding in the EBCOT and the DWT only to minimize the code modification. The maximum achievable speedup is limited by the sequentialization in this loop-level parallelization approach.

We analyze the whole code to investigate the existing concurrency in JPEG2000 and parallelize the level shift stage, the inter-component transform stage, the quantization stage, and a portion of the stream I/O routine in addition to the Tier-1 encoding and DWT stages. We also introduce a novel data decomposition scheme (discussed in Section 2) to achieve high performance while reducing the programming complexity, and we apply it to the multiple algorithmic kernels in JPEG2000.

The DWT is one of the most computationally intensive parts in JPEG2000 and also an important kernel in other application areas [11]. We provide a detailed analysis of its performance on the Cell/B.E. We optimize and tune the column grouping strategy based on our data decomposition scheme to increase spatial locality and design a new loop merging approach to increase temporal locality. In addition, we investigate the relative performance of the floating point operations and its fixed point approximation [1] on the Cell/B.E.

We achieve an overall speedup of 6.6 and 3.1 for lossless and lossy encoding with 8 SPEs compared to the single SPE performance. Also, our implementation obtains 6.9 and 7.4 times higher performance over the PPE-only case. The Cell/B.E. demonstrates 3.2 and 2.7 times faster encoding time relative to the Intel Pentium IV 3.2 GHz processor for the lossless and lossy cases, respectively. For the DWT, the Cell/B.E. outperforms the Pentium IV processor by 9.1 and 15 times for the lossless and lossy cases, respectively. We further test our implementation on a single IBM QS20 blade with two Cell/B.E. processors. The performance scales up to 16 SPEs, and this suggests that our implementation can benefit from the future Cell/B.E. processor, which may include 32 SPEs. Also, significantly higher performance is obtained in comparison with the previous Motion JPEG2000 encoder implementation [10] for the Cell/B.E. The source code of this project is freely available from our CellBuzz project in SourceForge (`http://sourceforge.net/projects/cellbuzz`).

This paper is organized as following: We describe our data decomposition scheme in Section 2. Parallelization and vectorization strategies of JPEG2000 are explained in Section 3 and Section 4, respectively. In Section 5, we provide detailed performance results, and finally, Section 6 concludes the paper.

2 Data Decomposition Scheme

Data layout is an important design issue in parallel programming. It is even more important for the Cell/B.E. due to the alignment and size requirements for DMA data transfer and SIMD load/store. DMA on the Cell/B.E. requires 1, 2, 4, 8 byte alignment to transfer 1, 2, 4, 8 bytes of data and 16 byte alignment to transfer a multiple of 16 bytes. DMA data transfer becomes most efficient if data addresses are cache line aligned in both main memory and the SPE Local Store, and data transfer size is an even multiple of the cache line size [5]. SIMD load/store instructions for vectorization also require quad-word data alignment.

Our data decomposition scheme, shown in Figure 1, satisfies the above requirements for the two dimensional array

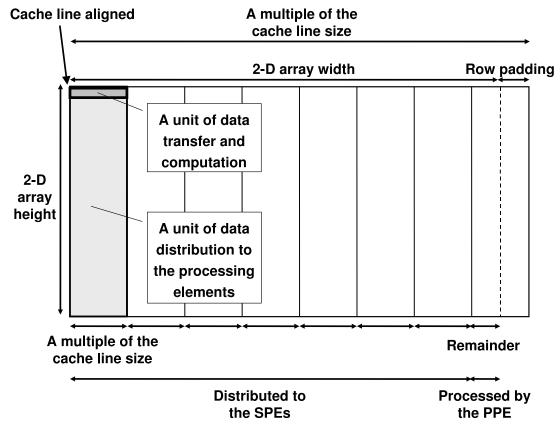
Cache line aligned
A multiple of the cache line size
2-D array width
Row padding
A unit of data transfer and computation
2-D array height
A unit of data distribution to the processing elements
A multiple of the cache line size
Remainder
Distributed to the SPEs
Processed by the PPE

Figure 1: Data decomposition scheme for two dimensional array

with an arbitrary width and height, assuming that every row can be arbitrarily partitioned into multiple chunks for independent processing. First, we pad every row to force the start address of every row to be cache line aligned. Then, we partition the data array to multiple chunks, and every chunk except for the last has the width a multiple of the cache line size. All the chunks have the height identical to the data array height. These data chunks become a unit of data distribution to the processing elements. The constant width chunks are distributed to the SPEs and the PPE processes the last remainder chunk. The SPE traverses the assigned chunks by processing every single row in the chunk as a unit of data transfer and computation.

This data decomposition scheme has several impacts on the performance and the programmability. First, every DMA transfer in the SPE becomes cache line aligned, and the transfer size becomes a multiple of the cache line size. This results in efficient DMA transfers and reduced programming complexity. If data are not properly aligned, or the data transfer size has an arbitrary value, additional programming would have been required to satisfy the conditions for correctness. Reduced programming complexity, or in other words, shorter code size also saves the Local Store space, and this is important for the Cell/B.E. since the Local Store size is relatively small. Also, under our data decomposition scheme, there is no cache conflict since every cache line in the data array is accessed either by the PPE or a DMA instruction issued by one SPE. The remainder chunk with an arbitrary width is processed by the PPE to enhance the overall chip utilization. Our data decomposition scheme also satisfies the alignment requirement for SIMD load/store.

Second, the Local Store space requirement becomes constant independent of the data array size. As mentioned above, a single row in the chunk, which has the constant width, becomes a unit of data transfer and computation in the SPE. This leads to a constant memory requirement, and we can easily adopt the optimization techniques that require additional Local Store space. For example, double buffering or multi-level buffering is an efficient technique for hiding latency but increases the Local Store space requirement at the same time. However, owing to the constant memory requirement in our data decomposition scheme, we can increase the level of buffering to a higher value that fits within the Local Store.

In addition, fixed data size leads to a constant loop count, which enables the compiler to better predict the program's runtime behavior. This helps the compiler to use optimization techniques such as loop unrolling, instruction rescheduling, and compile time branch prediction. Compile time instruction rescheduling and branch prediction compensate for the lack of runtime out-of-order execution and dynamic branch prediction support in the SPE.

3 Parallelization of JPEG2000

3.1 Parallelism in JPEG2000

The level shift, inter-component transform, and quantization stages are basically pixel-wise independent. Therefore, arbitrary partitioning is possible. Discrete Wavelet Transform (DWT) consists of two steps. First, the vertical filtering step partitions the original image to the low pass subband and the high pass subband in the vertical direction, and then the horizontal filtering step is followed to partition in the horizontal direction. In the vertical filtering, every column can be processed independently, and in the horizontal filtering, every row can be processed independently. For Tier-1 encoding in the EBCOT algorithm, an image array is partitioned to multiple code blocks, and every code block can be processed independently.

3.2 Parallelization Strategy

Figure 2 summarizes the work partitioning among the PPE and the SPEs for JPEG2000 encoding. The level shift, inter-component transform, DWT, Tier-1 encoding and quantization stages are fully parallelized using both the PPE and the SPEs. The level shift and inter-component transform stages are merged to minimize the data transfer. The read component data stage, which includes type conversion from the Jasper specific intermediate data type to four byte integer data type, is partially parallelized. We apply our data decomposition scheme to every parallelized stage except for the Tier-1 encoding stage. For the Tier-1 encoding stage, we adopt a work queue to distribute the workloads to the processing elements.

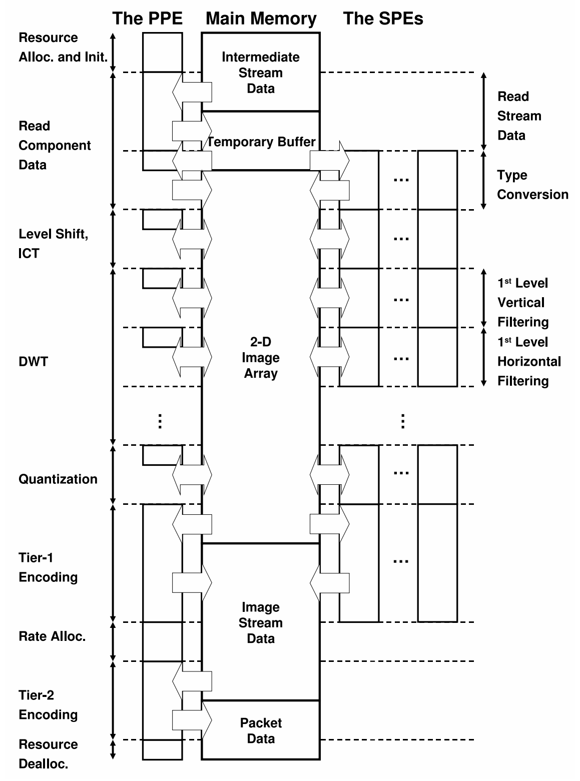

Figure 2: Work partitioning among the PPE and the SPEs for encoding

In Tier-1 encoding, efficient DMA data transfer is less important owing to the relatively high computation to communication ratio. Thus, we focus on increasing the overall chip utilization while minimizing the interaction among the processing elements. We use the PPE and SPE threads to encode the code blocks and maintain a work queue for load balancing. Note that the processing time for Tier-1 encoding is dependent on the input data characteristics, and we cannot achieve load balancing by merely distributing an identical number of code blocks to the processing elements. Both our implementation and the previous Motion JPEG2000 encoder [10] use a work queue for load balancing but use the PPE for different purposes. In [10], the authors support lossless encoding only and overlap the Tier-1 encoding stage with the Tier-2 encoding stage. In their implementation, the PPE (or the PPEs in their two Cell/B.E. implementations) performs Tier-2 encoding and the code block distribution to the SPEs, while only the SPE threads perform Tier-1 encoding. In the lossy encoding process, the rate control stage appears between the Tier-1 encoding stage and the Tier-2 encoding stage. This prevents the overlap in our implementation, and we use the PPE and SPE threads for Tier-1 encoding. Another distinction is the code block size. In [10], the authors select 32 by 32 pixels code block size instead of 64 by 64, the maximum code block size in the standard. Smaller code block size reduces the Local Store memory requirements and enables double buffering, but increases the interaction among the PPE and SPE threads. This lowers the scalability of the implementation.

Previous parallelization strategies for the DWT [3,11,15] suggest the trade-offs between computation and communication overheads for the different underlying architectures. Lifting based DWT is also proposed in [12], by which, in-place DWT can be performed faster than previous convolution based DWT [6]. Still, poor cache behavior in a column-major traversal in a C language implementation becomes a bottleneck in performance. Initially, a matrix transpose, which converts a column-major traversal to a row-major traversal, is adopted to improve cache behavior. Loop tiling, or column grouping, is introduced in [3] to improve cache behavior without matrix transpose. In [10], the authors parallelize convolution based DWT for the Cell/B.E. by partitioning the data array to 128 by 128 pixels tiles with the overlap among the adjacent tiles. The net tile size is 112 by 112 pixels. Their implementation does not satisfy the cache line alignment requirements for the most efficient DMA transfer due to the overlapped area. We optimize lifting based DWT for the Cell/B.E. For the horizontal filtering, we assign an identical number of rows to each SPE, and a single row becomes a unit of data transfer and computation. Every row is cache line aligned, and the data transfer size becomes a multiple of the cache line size owing to the row padding in our data decomposition scheme. For the vertical filtering, we tune the column grouping approach by fixing column group size to a multiple of the cache line size and partition the data array based on our data decomposition scheme. Our implementation enhances the DMA data transfer efficiency, which is essential to achieve high scalability.

4 Vectorization of JPEG2000

Based on our data decomposition scheme, we vectorize the level shift, inter-component transform, and quantization stages in a straightforward way. Yet, vectorization of the DWT involves interesting issues related to the Cell/B.E., and we discuss the issues in this section.

Instruction	Description	Latency
mpyh	two byte integer multiply high	7 cycles
mpyu	two byte integer multiply unsigned	7 cycles
a	add word	2 cycles
fm	single precision floating point multiply	6 cycles

Table 1: Latency for the SPE instructions

In [1], the authors suggest the fixed point representation for the real numbers in the JPEG2000 lossy encoding process to enhance the performance and the portability. The authors assume that fixed point instructions are generally faster than floating point instructions. However, the current version of the Cell/B.E. chip is optimized for (single precision) floating point operations, and the floating point instructions have comparable speed to the fixed point instructions. Moreover, the SPE instruction set architecture does not support four byte integer multiplication; thus four byte integer multiplication needs to be emulated by two byte integer multiplications and additions. Table 1 summarizes the latency for the two byte integer multiplications, four byte integer addition, and single precision floating point multiplication. Therefore, the fixed point representation loses its benefit on the Cell/B.E. We replace the fixed point representation in Jasper code with the floating point representation to achieve high performance.

```
Input: an image data array $array\_data$, number of rows $number\_of\_rows$, row width including padding $stride$
Output: an image data array $array\_data$
$high\_start \leftarrow number\_of\_rows/2$;
/* 1st lifting step for the vertical filtering */
$p\_low \leftarrow array\_data[0]$;
$p\_high \leftarrow array\_data[high\_start * stride]$;
$n \leftarrow high\_start - 1$;
for $i \leftarrow 0$ to $n - 1$ do
    $*p\_high \leftarrow *p\_high - ((*p\_low + *(p\_low + stride))/2)$;
    $p\_low \leftarrow p\_low + stride$;
    $p\_high \leftarrow p\_high + stride$;
end
$*p\_high \leftarrow *p\_high - *p\_low$;
/* 2nd lifting step for the vertical filtering */
$p\_low \leftarrow array\_data[0]$;
$p\_high \leftarrow array\_data[high\_start * stride]$;
$*p\_low \leftarrow *p\_low + ((*p\_high + 1)/2)$;
$p\_low \leftarrow p\_low + stride$;
$n \leftarrow high\_start - 1$;
for $i \leftarrow 0$ to $n - 1$ do
    $*p\_low \leftarrow *p\_low + ((*p\_high + *(p\_high + stride) + 2)/4)$;
    $p\_low \leftarrow p\_low + stride$;
    $p\_high \leftarrow p\_high + stride$;
end
```

Algorithm 1: Pseudo code for original implementation

The DWT algorithm consists of multiple steps: one splitting step and two lifting steps in the lossless mode and one splitting step, four lifting steps, and one optional scaling step in the lossy mode. Considering that the entire column group data for a large image does not fit into the Local Store, 3 or 6 steps in the vertical filtering involve 3 or 6 DMA data transfer of the entire column group data. As

```
Input: an image data array $array\_data$, number of rows $number\_of\_rows$, row width including padding $stride$
Output: an image data array $array\_data$
$high\_start \leftarrow number\_of\_rows/2$;
/* interleaves 1st and 2nd lifting steps for the vertical filtering */
$p\_low \leftarrow array\_data[0]$;
$p\_high \leftarrow array\_data[high\_start * stride]$;
$*p\_high = *p\_high - ((*p\_low + *(p\_low + stride))/2)$;
$*p\_low = *p\_low + ((*p\_high + 1)/2)$;
$p\_low \leftarrow p\_low + stride$;
$p\_high \leftarrow p\_high + stride$;
$n \leftarrow high\_start - 2$;
for $i \leftarrow 0$ to $n - 1$ do
    $*p\_high \leftarrow *p\_high - ((*p\_low + *(p\_low + stride))/2)$;
    $*p\_low \leftarrow *p\_low + ((*(p\_high - stride) + *p\_high + 2)/4)$;
    $p\_low \leftarrow p\_low + stride$;
    $p\_high \leftarrow p\_high + stride$;
end
$*p\_high \leftarrow *p\_high - *p\_low$;
$*p\_low \leftarrow *p\_low + ((*(p\_high - stride) + *p\_high + 2)/4)$;
```

Algorithm 2: Pseudo code for our interleaved implementation

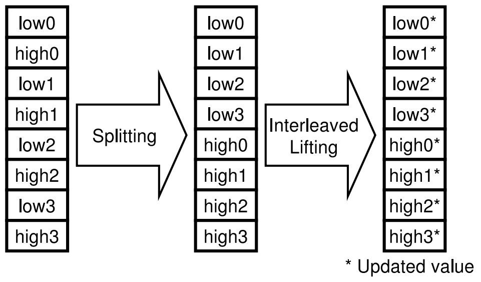

Figure 3: The splitting step and the interleaved lifting step for the vertical filtering

the number of SPEs increases, the limited off-chip memory bandwidth becomes a bottleneck and nullifies the performance enhancement achieved by vectorization. To reduce the amount of DMA data transfer, we interleave the lifting steps. Algorithm 1 illustrates the original algorithm for the lossless mode assuming that the number of rows is even and larger than four. By analyzing the data dependency in Algorithm 1, we notice that two lifting steps can be interleaved. Algorithm 2 depicts our interleaved algorithm. The splitting step can also be merged with the next two lifting steps. Figure 3 illustrates the splitting step and the interleaved lifting step in the lossless vertical filtering. Initially, the low and high pass components are interleaved in the input array, and the splitting step separates the low and high pass components. This splitting step involves the DMA data transfer of the whole column group data. If we adjust the pointer addresses for the low and high pass components and increase the increment size in the next interleaved lifting step, the splitting step can be merged with the interleaved lifting step. This merges three steps in the lossless mode to a single step and reduces the amount of data transfer. However, updating the high pass part in the merged single step will overwrite the input data before it is read, and this leaves us with a problem. In Figure 3, *high0* and *low0* are updated first in the interleaved lifting step, and *high1* and *low1* are updated next. If we adjust the input data pointer and skip the splitting step, updating *high0* overwrites *low2* in the input data array before it is read. To remedy this problem, we use an auxiliary buffer (in main memory), and the updated high pass data are written to the buffer first and copied to the original data array after the single merged step is finished. The amount of data transfer related to the auxiliary buffer is half of the entire column group, and this halves the amount of data transfer for the splitting step. We recently find that a similar idea is published in [7] for the lossy case. Loop fusion for the four lifting steps is described in the paper. By combining this idea with our approach, we merge one splitting step, four lifting steps, and the optional scaling step in the lossy mode into a single loop to further reduce the DMA bandwidth requirement.

5 Performance Results

We use the *gcc* compiler in the Cell SDK 2.1 with *-O5* optimization flag for the performance analysis. Our baseline code is the open source Jasper 1.900.1 (`http://www.ece.uvic.ca/~mdadams/Jasper`), and the test file is a 28.3 MB 3800 × 2600 color image (`http://www.jpeg.org/jpeg2000guide/testimages/WalthamWatch/Original/waltham_dial.bmp`). The default option and *-O mode=real -O rate=0.1* are applied for lossless and lossy encoding, respectively. We transcode the image from the BMP format to the JPEG2000 format and analyze the JPEG2000 encoding time. The BMP decoding time is disregarded. Our experiments use real hardware, the IBM QS20 Cell blade server with dual Cell/B.E. 3.2 GHz chips (rev. 3) and 1 GB main memory.

5.1 The Encoding Time and the Scalability

Figures 4 and 5 display the execution time and the speedup for lossless and lossy encoding, respectively. Considering that the EBCOT algorithm is branchy and integer based, the PPE runs the code faster than the SPE for Tier-1

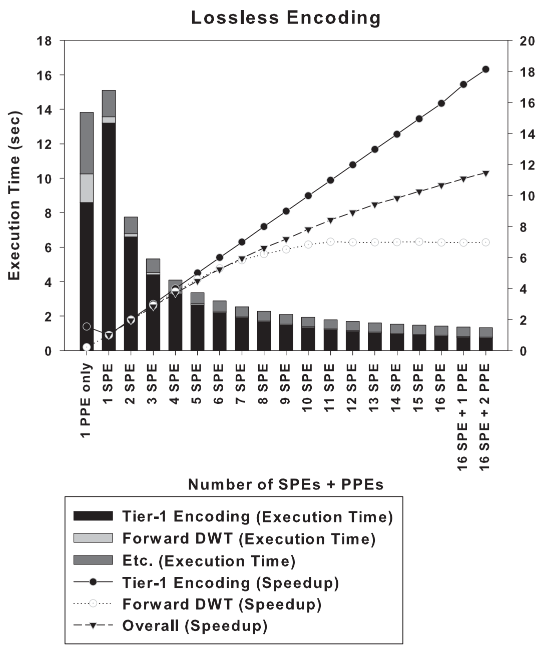

Figure 4: The execution time and the speedup for lossless encoding. Additional PPEs participate in Tier-1 encoding.

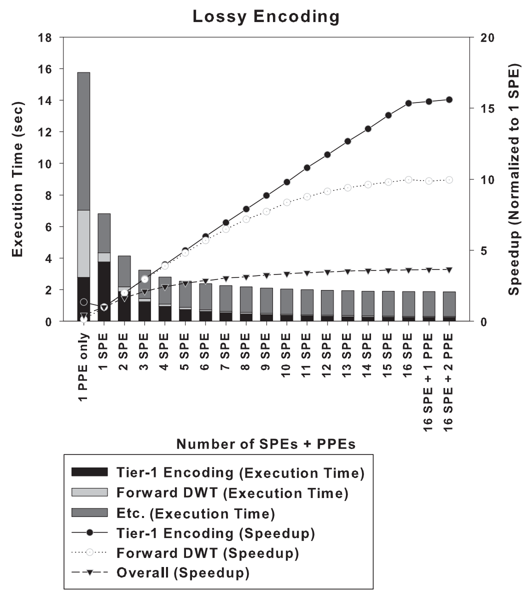

Figure 5: The execution time and the speedup for lossy Encoding. Additional PPEs participate in Tier-1 encoding.

encoding. Therefore, 1 PPE only case outperforms the single SPE performance. Still, we achieve near linear speedup with the increasing number of SPEs and the extra speedup with the additional PPE threads. As a consequence, 16 SPE + 2 PPE case completes Tier-1 encoding significantly faster than 1 PPE only case.

In the case of the DWT, 1 SPE case outperforms 1 PPE only case by far, and we demonstrate a remarkable speedup with additional SPEs. vectorization contributes to the superb single SPE performance, and in the lossy encoding case, execution time is further reduced by replacing the fixed point representation with the floating point representation. The efficient use of the off-chip memory bandwidth based on our data decomposition scheme and the reduced bandwidth requirement owing to the loop interleaving realize the high scalability.

Overall lossless encoding demonstrates a parallel speedup while the lossy encoding performance flattens with the increasing number of SPEs due to the sequential rate allocation stage, which takes around 60% of the total execution time in 16 SPE + 2 PPE case.

5.2 Comparison with the Previous Implementation

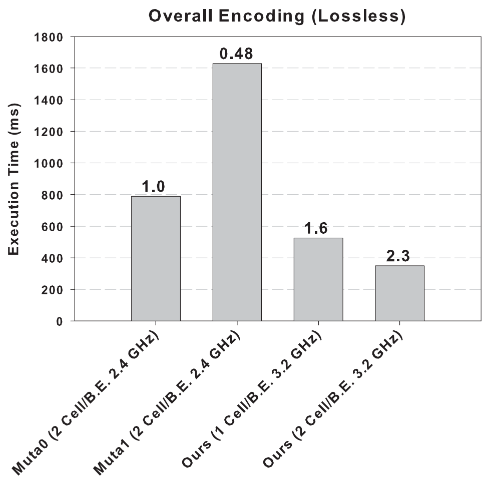

Figure 6: The overall performance comparison with the previous implementations for the Cell/B.E. The numbers above the bars denote the speedup relative to Muta0.

Figures 6, 7, and 8 summarize the performance comparison. Muta0 and Muta1 in the figures denote the implementations in [10], where the Motion JPEG2000 encoding algorithm is optimized for the Cell/B.E. In Muta0, two encoding threads encode two different frames concurrently with two Cell/B.E. processors. The two chips work in a synergistic way to increase the overall throughput. In Muta1, one encoding thread encodes all the frames using two Cell/B.E. processors.

We use the performance numbers reported by the authors for the comparison. The authors exclude the Motion JPEG2000 format building time and measure the encoding

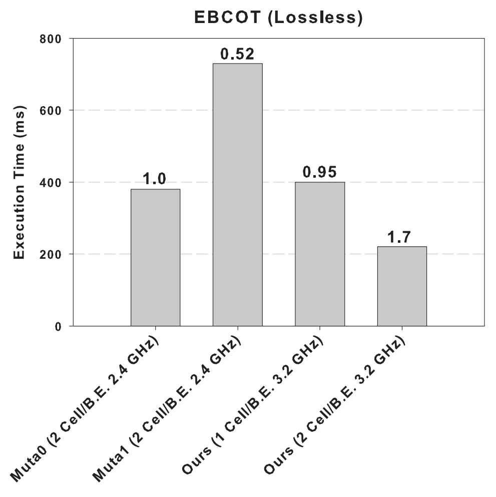

Figure 7: The EBCOT (Tier-1 + Tier2) encoding performance comparison with the previous implementations for the Cell/B.E. The numbers above the bars denote the speedup relative to Muta0.

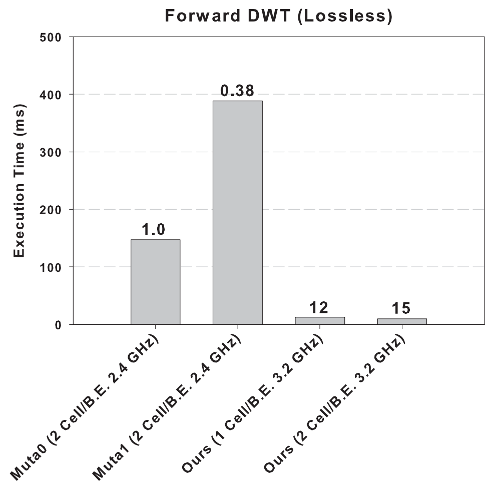

Figure 8: The DWT performance comparison with the previous implementations for the Cell/B.E. The numbers above the bars denote the speedup relative to Muta0

time for 24 frames with the size of 2048×1080. Total execution time is divided by the number of encoded frames to compute the per frame encoding time. In Muta0, the encoding time for one frame can be up to two times higher than the reported number. We scale down the test image for our implementation to 2048×1080 for the fair comparison.

Still, there are multiple caveats in the comparison. First, the Cell/B.E. 2.4GHz, instead of the Cell/B.E. 3.2 GHz, is used in [10]. Second, our implementation based on Jasper includes the reading and type conversion time from the Jasper specific intermediate stream, JPEG2000 format building time, and the final file I/O time to save the encoded file. This may be different from [10]. Third, even though the test images have the identical size, different characteristics of the images may affect the result.

Our implementation with one Cell/B.E. processor and two Cell/B.E. processors demonstrates superior overall performance than the previous implementations with the two Cell/B.E. processors. This is mainly due to the following reasons. First, our EBCOT implementation reveals higher scalability than the previous implementation. Minimized communication among the PPE and the SPEs enhances the scalability in addition to the implementation details. Second, adopting a lifting based scheme instead of a convolution based scheme, combined with the higher chip clock frequency, leads to the higher single SPE performance for the DWT, and our data decomposition scheme and the loop interleaving contribute to the higher scalability. Third, we parallelize the level shift, inter-component transform, and quantization stages while these stages are executed on the PPE in [10] to avoid the offloading overhead. The offloading overhead is insignificant based on our data decomposition scheme.

5.3 Comparison with the Intel x86 Architecture

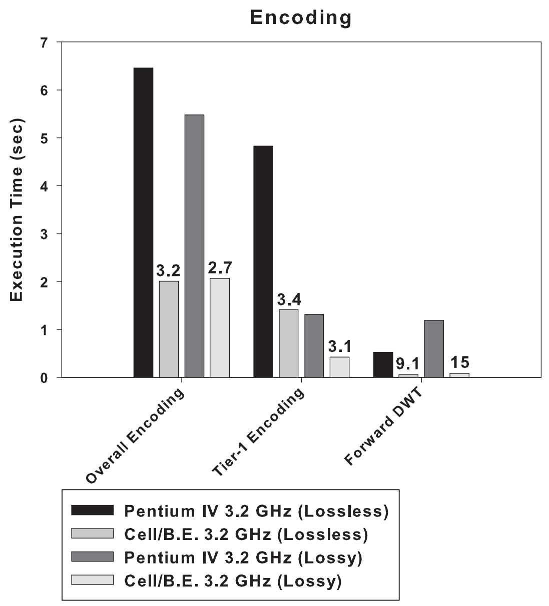

Figure 9: The encoding performance comparison of the Cell/B.E. to that of the Intel Pentium IV 3.2 GHz processor. The numbers above the bars denote the speedup relative to the Intel Pentium IV processor.

Figure 9 summarizes the performance comparison between the Cell/B.E. and the Intel Pentium IV 3.2 GHz with

2 MB cache memory and 2 GB main memory. Jasper code on the Pentium IV processor is compiled with the *gcc* version 4.1.2 and *-O5* optimization flag (instead of the default *-g -O2*). To make the comparison as fair as possible, we apply the optimizations except for parallelization, vectorization, and other optimizations specific to the Cell/B.E. to both architectures. Note that the Pentium IV processor also supports SIMD instructions, but vectorization is not implemented in the Jasper code for the Pentium IV processor. Also, for lossy encoding, the Cell/B.E. performs the floating point arithmetic while the Pentium IV processor emulates the floating point operations with the fixed point instructions. The Cell/B.E. outperforms the Pentium IV in the comparison. Especially, the Cell/B.E. boasts the impressive performance for the DWT while the sequential part of the code running on the PPE lowers the overall speedup. These performance numbers show that the Cell/B.E. has superb performance for the floating point based loop intensive algorithms, and its relatively low single core performance for the branchy and integer based algorithms can be compensated by exploiting multiple SPEs.

6 Conclusion

We analyze the underlying parallelism in JPEG2000 and design new high-performance parallelization strategies including the novel data decomposition scheme. Our data decomposition scheme enhances the performance and reduces the programming complexity at the same time. The efficiency of the approach is demonstrated by the experimental results. In addition to the superior performance within a single Cell/B.E., our implementation shows the scalable performance on a single blade with two Cell/B.E. processors. This suggests that our approach will work efficiently even in the future Cell/B.E. processors with more SPEs. In conclusion, to unveil the Cell/B.E.'s full performance, judicious implementation strategy is required. Yet, under the proper implementation strategy, the Cell/B.E. can unleash superb performance, especially for the floating point based loop intensive algorithms such as the DWT.

Acknowledgments

This work was supported in part by an IBM Shared University Research (SUR) award and NSF Grants CNS-0614915 and CAREER CCF-0611589. We acknowledge Georgia Institute of Technology, its Sony-Toshiba-IBM Center of Competence, and the National Science Foundation, for the use of Cell Broadband Engine resources that have contributed to this research.

References

[1] M. D. Adams and F. Kossentini. Jasper: a software-based JPEG-2000 codec implementation. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, volume 2, pages 53–56, Vancouver, BC, Canada, Sep. 2000.

[2] P. J. Burt and E. H. Anderson. The Laplacian pyramid as a compact image code. *IEEE Trans. Commun.*, 31(4):532–540, Apr. 1983.

[3] D. Chaver, M. Prieto, L. Pinuel, and F. Tirado. Parallel wavelet transform for large scale image processing. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2002)*, pages 4–9, Ft. Lauderdale, FL, USA, Apr. 2002.

[4] ISO and IEC. *ISO/IEC 15444-1: Information technology-JPEG2000 image coding system-part 1: Core coding system*, 2000.

[5] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26(3):10–23, 2006.

[6] D. Krishnaswamy and M. Orchard. Parallel algorithms for the two-dimensional discrete wavelet transform. In *Proc. Int'l Conf. Parallel Processing*, volume 3, pages 47–54, 1994.

[7] R. Kutil. A single-loop approach to simd parallelization of 2-D wavelet lifting. In *Proc. Euromicro Int'l Conf. on Parallel, Distributed, and Network-Based Processing (PDP 2006)*, Montbeliard, France, Feb. 2006.

[8] C.-J. Lian, K.-F. Chen, H.-H. Chen, and L.-G. Chen. Analysis and architecture design of block-coding engine for EBCOT in JPEG 2000. *IEEE Trans. Circuits and Systems*, 13(3):219–230, Mar. 2003.

[9] P. Meerwald, R. Norcen, and A. Uhl. Parallel JPEG2000 image coding on multiprocessors. In *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS 2002)*, pages 2–7, Ft. Lauderdale, FL, USA, Apr. 2002.

[10] H. Muta, M. Doi, H. Nakano, and Y. Mori. Multilevel parallelization on the Cell/B.E. for a Motion JPEG 2000 encoding server. In *Proc. ACM Multimedia Conf. (ACM-MM 2007)*, Augsburg, Germany, Sep. 2007.

[11] O. Nielsen and M. Hegland. Parallel performance of fast wavelet transform. *International Journal of High Speed Computing*, 11(1):55–73, Jun. 2000.

[12] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2):511–546, Mar. 1998.

[13] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Trans. Image Processing*, 9(7):1158–1170, Jul. 2000.

[14] P. P. Vaidyanathan. Quadrature mirror filter banks, m-band extensions, and perfect reconstruction techniques. *IEEE ASSP Magazine*, 4(7):4–20, Jul. 1987.

[15] L. Yang and M. Misra. Coarse-grained parallel algorithms for multi-dimensional wavelet transforms. *J. Supercomputing*, 12(1-2):99–118, Jan. 1998.