Optimizing Discrete Wavelet Transform on the Cell Broadband Engine

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

Introduction

Discrete Wavelet Transform (DWT) is one of the most computationally expensive algorithmic kernels in JPEG2000 and also an important algorithm in other application areas [1]. The DWT algorithm has abundant parallelism to be exploited by multicore processors. Yet, this algorithm has distinct memory access patterns in horizontal and vertical filtering, and high bandwidth requirements, which become a performance bottleneck in optimization.

The Sony-Toshiba-IBM Cell Broadband Engine (Cell/B.E.) is a heterogeneous multicore processor with the unique memory subsystem. The Cell/B.E.'s novel architecture raises an implementation challenge with potentially high pay-offs in performance.

We provide the optimization techniques for the Cell/B.E. to fully utilize this unique and highly capable processor, and overcome the performance bottleneck. We also provide a performance comparison with the AMD Barcelona (Quad-core Opteron) processor to highlight the advantage of the Cell/B.E.'s architecture over general purpose multicore processors in processing regular and bandwidth intensive applications.

Cell Broadband Engine

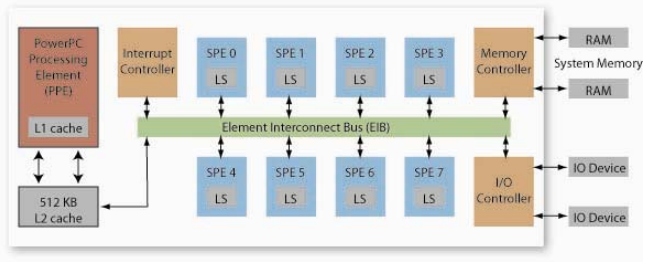

Figure 1: The Cell Broadband Engine architecture

The Cell/B.E. consists of two types of cores: one PPE and eight SPEs. The SPEs' have unique and simple architecture distinguishes the Cell/B.E. from other multicore processors. The SPE is an in-order SIMD accelerator with local memory (Local Store) and does not support dynamic branch prediction or scalar instructions. This significantly reduces the transistor counts and power consumption of the core. Therefore, a single chip can include more cores within transistor or power budget. Moreover, the Cell/B.E. has highly scalable memory subsystem. The SPE has two mechanisms for data movements. Cache coherent DMA messages provide a mechanism for the data transfers among the main memory and the Local Stores. The load/store instructions move data from the SPE's Local Store to its registers or from the registers to the Local Store. These load/store instructions are local to the SPE and do not generate cache coherency messages. This reduces cache coherency traffic and enhances the scalability of the architecture with the increasing number of cores. Additional number of cores and the highly scalable architecture provide great potential for high performance. Especially, if an application has regular execution patterns that can be accurately predicted in compile time, additional cores can contribute to high aggregate computing power without noticeable sacrifice in a single core performance. A programmer or the compiler can provide branch hints instead of relying on dynamic branch prediction, and the compiler can reschedule the instructions to compensate the lack of dynamic out-of-order execution support. We can expect an additional performance boost, if an application code can be efficiently vectorized using SIMD instructions. Explicit data transfer requires additional programming but at the same time, provides an opportunity for the fine grain optimization specific to the application requirements.

Optimization Approaches

The DWT algorithm has two distinctive memory access patterns. The algorithm scans a 2-D image array in row major order for the horizontal filtering, and column major order for the vertical filtering. For C implementation, bad cache behavior in column major traversal decreases the performance significantly, and column grouping technique [2], which groups and interleaves multiple column major traversals to enhance the cache behavior, is adopted to resolve this problem. We designed data decomposition scheme for the Cell/B.E., and this scheme can be combined with the column grouping approach for the efficient data transfer between main memory and the Local Store.

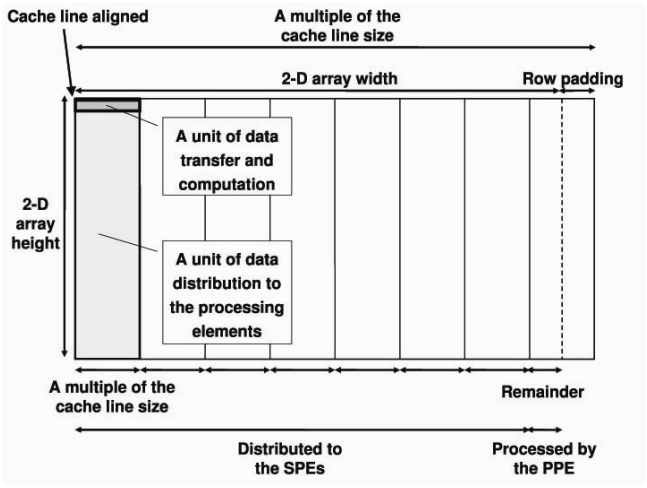

Figure 2: Data decomposition scheme for 2-D array

Figure 2 displays the data decomposition scheme for the two dimensional array with an arbitrary width and height, assuming that every row can be arbitrarily partitioned to multiple chunks for independent processing. The efficient

Disclaimers and footnotes.

data transfer and vectorization on the Cell/B.E. require proper data alignment, and our data decomposition scheme addresses these alignment issues for the two dimensional array with the above assumptions. Row padding forces the start address of every row to be cache line aligned and the length of every row to be a multiple of the cache line size. Data array is partitioned to the multiple chunks with a constant width (a multiple of the cache line size) and the remainder chunk with an arbitrary width. The SPEs process the constant width data chunks and the PPE processes the remainder chunk. A single row in the data chunk becomes a unit of data transfer and processing for the SPEs.

This data decomposition scheme has the following impacts in 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. Therefore, we can expect the highly efficient DMA transfer and the reduced programming complexity. With the unaligned data or the arbitrary data transfer size, we need an additional programming to satisfy the conditions for the correct and efficient data transfer. Similar analysis applies to the vectorization. Second, we can easily adopt the optimization techniques which require additional Local Store space. 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. Thus, the Local Store space requirement becomes constant independent of the data array size, and we can exactly estimate the memory requirements in static time. In addition, fixed data size leads to a constant loop count, and if a loop count is constant, the compiler can better predict the runtime behavior. This facilitates the compiler optimization techniques such as loop unrolling, instruction rescheduling, and compile time branch prediction.

We optimized the DWT algorithm in Jasper [3], a reference JPEG2000 library implementation, and the following summarizes our optimization approaches based on the data decomposition scheme.

- **Parallelization and Vectorization**
- **Real number representation** – Replaced fixed point real number representation (assign fixed number of bits for fractional part, and the remaining bits for the integer part) with floating point representation to exploit the Cell/B.E.'s superb single precision floating point performance.
- **Loop interleaving** – interleaves the loops for multiple lifting steps and a splitting step in the algorithm to reduce the bandwidth requirements.
- **Multi-level buffering**
- **Fine grain data transfer control** – Explicit data transfer on the Cell/B.E. enables fine grain data transfer control to overlap data transfer with computation in addition to multi-level buffering.

Performance Evaluation

We compare the Cell/B.E. 3.2 GHz processor's performance with the AMD Barcelona 2.0 GHz (Quad-Core Opteron Processor 8350). We optimize the code for Barcelona using PGI C compiler, a parallel compiler for multicore optimization, and user provided compiler directives. Table 1 summarizes the optimization for the Barcelona.

Table 1: Optimizations for the Barcelona

Parallelization	OpenMP based parallelization
Vectorization	Auto-vectorization with compiler directives for pointer disambiguation
Real number representation	Identical to the Cell/B.E. case
Loop interleaving	Identical to the Cell/B.E. case
Run-time profile feedback	Compile with the run-time profile feedback (-Mpfo)

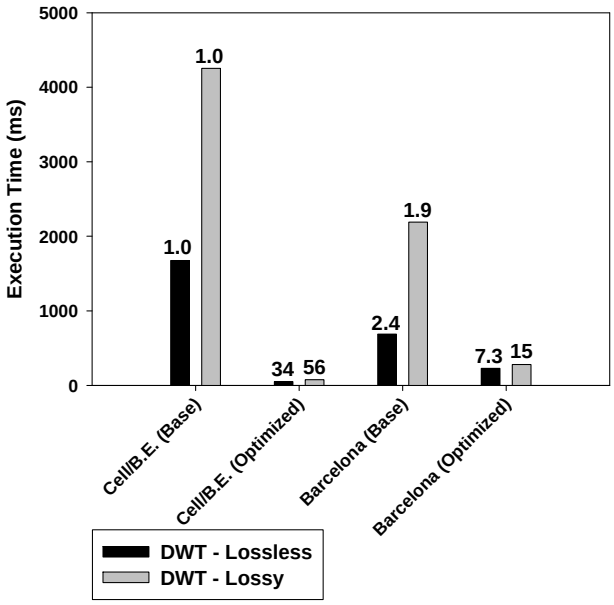

Figure 3: Experimental Results. The numbers above the bars denote the speedup relative to the Cell/B.E. (Base).

Figure 3 shows that the baseline implementation for the Cell/B.E. (running on the PPE only) has lower performance, but this processor's unique architecture, combined with the judicious optimization approaches, has a great potential for exceptional performance to pay-off its higher programming complexity.

References

[1] O.M. Nielsen and M. Hegland, "Parallel performance of fast wavelet transform", International Journal of High Speed Computing, Vol 40, No. 1, Jun. 2000.

[2] D. Chaver, M. Prieto, L. Pinuel, and F. Tirado, "Parallel wavelet transform for large scale image processing", In Proc. of Int'l Parallel and Distributed Processing Symp., pages 4–9, Apr. 2002.

[3] M. D. Adams and F. Kossentini, "Jasper: a software-based JPEG-2000 codec implementation", In Proc. of Int'l conf. on image processing, volume 2, pages 53–56, Sep. 2000.