

IBM Research Report

Digital Media Indexing on the Cell Processor

Lurng-Kuo Liu¹, Qiang Liu², Apostol (Paul) Natsev¹, Kenneth A. Ross^{1,3},
John R. Smith¹, Ana Lucia Varbanescu^{1,4}

¹IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

²IBM Research Division
China Research Laboratory
Building 19, Zhouguancun Software Park
8 Dongbeiwang West Road, Haidian District
Beijing, 100094
P.R.C.

³Columbia University

⁴Delft University of Technology



Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

DIGITAL MEDIA INDEXING ON THE CELL PROCESSOR

Lurng-Kuo Liu¹, Qiang Liu², Apostol (Paul) Natsev¹, Kenneth A. Ross^{1,3}, John R. Smith¹ and Ana Lucia Varbanescu^{1,4}

¹IBM T. J. Watson Research Center, ²IBM China Research Lab
³Columbia University, ⁴Delft University of Technology

ABSTRACT

We present a case study of developing a digital media indexing application, code-named MARVEL, on the STI Cell Broadband Engine (CBE) processor. There are two aspects of the target application that require significant computing power: image analysis for feature extraction, and Support Vector Machine (SVM) based pattern classification for concept detection. We discuss the mapping of a large application like MARVEL onto a multicore processor, and show how feature extraction and concept detection can be implemented on the CBE. We discuss how the synergistic processing units of a CBE can be used to gain dramatic performance improvements. The empirical results of our experiments, conducted on a Cell blade running at 3.2 GHz, show that the CBE provides a significant performance speed-up in our digital media indexing application.

1. INTRODUCTION

Until recently, the primary performance gains for conventional processors were obtained by increasing the clock frequency. This technique has reached a point of diminishing returns — and even negative returns if power is taken into account. In response to this trend, multicore processors, also known as Chip multiprocessors (CMPs), have become more prevalent in vendors' solutions [5].

The Cell Broadband Engine (CBE) processor, jointly designed by Sony, Toshiba, and IBM, is a state-of-the-art multicore processor with nine processing elements operating on shared, coherent memory. It is the heart of the new Sony PlayStation 3, and also a building block for large high performance computing clusters.

Multicore processors bring a shift of paradigm in applications development. With multicore processors, the application development process must be oriented towards fully exploiting the available parallelism. Multimedia applications — due to their inherent data parallelism — are a prime target to explore in the context of this paradigm shift.

In this paper, we focus on the implementation of a multimedia content analysis and retrieval application, code-named MARVEL, on the CBE processor.

Our performance baseline is the performance of the original version of MARVEL running on a commodity architecture, such as the Intel Centrino processor¹. We describe how we ported MARVEL to the CBE processor, and how we optimized the code for the CBE architecture. We measure the performance of a CBE-optimized version of MARVEL running on a CBE processor. We demonstrate substantial performance improvements that highlight the strengths of the CBE processor for this type of workload.

2. MARVEL

Fueled by the rapid expansion of broadband Internet connectivity and increasing interest in online multimedia-rich applications, the volume of digital multimedia content has skyrocketed. The media and entertainment business is being changed by the deluge of digital media content. This deluge is driving the need for more effective methods for indexing, searching, categorizing and organizing the information. Manual processes that create metadata for indexing purposes cannot keep up with the explosion of rich media content. Manual annotation and cataloging are costly, time consuming, and often subjective — leading to incomplete and inconsistent annotations and poor system performance. New technologies are needed for reducing annotation costs.

MARVEL is one such technology developed at IBM Research. It uses multi-modal machine learning techniques to bridge the semantic gap for multimedia content analysis and retrieval [1,2,3,4]. MARVEL automatically annotates images and videos by recognizing the semantic entities — scenes, objects, events, people — that are depicted in the content. The MARVEL multimedia analysis engine applies machine learning techniques to model semantic concepts in images and video from automatically extracted visual

¹ Since we did not perform Intel-specific optimizations (such as the use of SSE), our results should not be interpreted as a direct comparison of the two processors.

descriptors. It automatically assigns labels (with associated confidence scores) to unseen content to reduce manual annotation load and improve searching, filtering, and categorization capabilities. The MARVEL multimedia retrieval engine integrates multimedia semantics-based searching with other search techniques (speech, text, metadata, audio-visual features, etc.), and combines content-based, model-based, and text-based retrieval for more effective image and video searching [2,4]. The computational cost of the semantic concept detection process and the need for scalability make the CBE very well-suited for use in digital media indexing systems.

We focus on the multimedia analysis aspects of MARVEL, and in particular, on optimizing semantic concept detection from images. This process involves extracting visual features (e.g., colors, textures, edges) from the content, and then evaluating statistical concept models (e.g., Support Vector Machine models) on these features.

3. THE CELL BROADBAND ENGINE PROCESSOR

The CBE has nine cores: one 64-bit Power Processor Element (PPE) and eight specialized Synergistic Processing Elements (SPE's). The PPE is a traditional 64-bit PowerPC (PPC) processor core with a VMX unit, 32KB L1 instruction cache, 32KB L1 data cache, and 512KB L2 cache. The PPE is the main processor of the Cell BE, being responsible for running the operating system and coordinating the SPEs. The SPEs cores are RISC-style processors designed for high-performance on streaming and data-intensive computation. An SPE has no cache, but it includes a 256 KB local store (LS) memory available for both instructions and data. Each SPE has 128 128-bit Single-Instruction Multiple-Data (SIMD) registers. All SPU instructions are inherently SIMD operations that can run at a variety of different granularities.

The PPE launches a thread for each SPE. Once the thread is launched, the SPE executes independently of the PPE, unless interaction or synchronization is required. The SPEs cannot access the main memory directly. In order to bring data into its LS or write results back to main memory, it uses asynchronous DMA transfers. The control flow between the PPE and the SPEs can also work via DMA transfers, but more convenient alternatives, such as mailboxes, signals, and interrupts, are also provided.

The CBE can obtain speed-up in two different ways: (1) by fully exploiting (in parallel) the PPE and eight SPEs, and (2) by specific optimizations (like vectorization) applied on the code running on each SPEs. The SPEs can achieve extremely good SIMD performance for integer and single-precision floating point operations when compared with traditional processors. An SPE can often execute two SIMD operations per cycle. By comparison, a Pentium 4 with SSE3 can only execute one SIMD operation every two cycles.

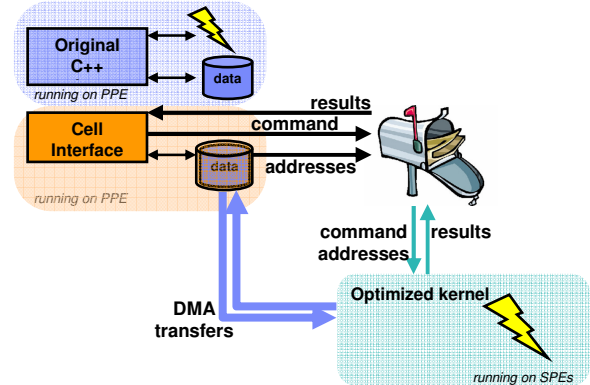


Fig. 1. SPE code integration in the PPE application

4. MAPPING MARVEL TO THE CBE

Marvel is implemented as a C++ application, with more than 50 classes and over 20,000 lines of code. Given its complexity, the porting operation had two main goals: (1) to reuse as much code as possible and (2) to have a working version of the application at all times. The step-by-step approach we have used is presented below, and it is suitable for any complex C++ application that has to be ported quickly to the CBE.

The first step is to port the application to the PPE, a phase that was straightforward. Starting from the Linux version of MARVEL, we were able to compile the entire application for the PPE in one day.

Next, we profiled the application to identify its most time-consuming modules. These kernels become the best candidates to be ported on the SPEs. For MARVEL, we identified five such kernels: the four feature extraction algorithms and the SVM-based classification. Together, these accounted for 85-90% of the time, assuming MARVEL did not resample images to lower resolution.

Once identified, the time-consuming kernels were re-implemented in C and optimized for running on the SPEs. To increase productivity, we isolated the kernels (in terms of functionality) and implemented them separately, focusing on getting the highest possible performance (see Section 5 for more details on these optimizations). Note that we did not alter MARVEL's control flow (for example, by processing many images with one function call).

To reintegrate the optimized kernels into the main application, we used a generic technique that allows the PPE to trigger and control the execution of the SPE code. On the PPE side, we provided a generic *cell interface* class that spawns the SPE thread and manages its execution. On the SPE side we created a *template* that provides the SPE-to-PPE interface (the same for all kernels), and allows for easy plug-in of the functional part of the kernel. The architecture is described in Figure 1.

All of the control flow between the PPE and SPE is done via mailboxes. All of the data required by the SPE processing is transferred via DMA. Double- or triple-buffering in the SPE code can overlap some of the DMA latency with computation. The porting effort, including the development of CBE-optimized algorithms, took about 4 person-months of effort.

5. OPTIMIZED ALGORITHMS ON THE SPE

For the current implementation, we focus on the four most time-expensive features that Marvel extracts from images: a *color-histogram (CH)*, a *color-correlogram (CC)*, an *edge-histogram (EH)*, and a *texture feature(TX)*.

Each extracted feature is represented as a vector of numbers that can be evaluated against a model using standard Support Vector Machine (SVM) testing functions. One model can evaluate one or several image features, and it contains a collection of vectors for each of these. MARVEL uses *precomputed* models of image content, models that are built by training an SVM classifier using a large corpus of labeled images.

The four feature-extraction algorithms and the SVM testing algorithm were re-implemented and hand-optimized for the Cell SPE. The algorithms were vectorized to take advantage of the SPE’s SIMD operations. In some cases, data is staged to the SPE in fragments so that images larger than the local store could be processed. Vectorization was done at as fine a granularity as possible. Many pixel operations used vectors consisting of sixteen 8-bit values, while floating-point operations used four 32-bit values.

5.1. Histogram Construction

Since pixels are independent for histogram construction, one can compute an image histogram for an arbitrarily large image by processing it one piece at a time. The image is read (using DMAs) in fragments, in separate R, G, and B arrays. The algorithm maps pixels to one of 166 buckets based on quantized HSV values derived from the original RGB values. We were able to identify the correct HSV-bucket without doing a full RGB-to-HSV conversion. Bucket determination uses 8-bit SIMD operations almost exclusively. Further, the code was written without any conditional tests, and all for-loops were of constant depth. As a result, the compiler was able to generate very efficient code that avoided the relatively high cost of a mispredicted branch on the Cell architecture. Once the histogram is computed, it is written to global memory using a DMA. In Marvel, the histogram is a vector containing 166 32-bit integer counts.

5.2. Correlogram

The correlogram code first converts the RGB image into a representation in which each pixel is associated with its quantized HSV value (a number between 0 and 165, as

described for histograms above). The correlogram feature tries to quantify, over the whole image, the degree of clustering among pixels with the same quantized HSV value. For each pixel P, one counts how many pixels there are within a square window around P having exactly the same pixel value as P. For our implementation, the length of the square was 17 pixels, meaning that pixels within 8 units (both horizontally and vertically) are considered.

SIMD instructions on the SPE are used to exploit the data parallelism. Contiguous 8-bit pixel values within a window are loaded into a SIMD register, and are processed together for all the windows to which they contribute. As a result, the SPE with SIMD performs very efficiently using data in registers most of the time.

5.3. Edge Histogram

For edge detection, the RGB image is iteratively processed by a sequence of filters, resulting in two new “images” containing the edge angle and edge magnitude for each pixel of the initial image.

The RGB image is read (using DMA) in slices. Each slice is processed to determine the gray value for each pixel, and then discarded, while the grayscale image is saved as a temporary result. Next, the image is processed for edge detection by applying two Sobel filters (horizontal and vertical). These are two 3x3 convolution filters (i.e., one pixel’s value is computed as a weighted sum of itself and its neighboring pixels) that detect the position of the edges in the image, generating two new images (Sobel-X and Sobel-Y). The convolution filters require successive data chunks to overlap by one line. To simplify the address computation for the DMA accesses, we have opted to read the grayscale image line by line, keeping only three lines active for the current processing.

Finally, the Sobel-X and Sobel-Y images are both used to compute the edge magnitude (M) and edge angle (Θ). The computation is performed pixel-by-pixel. If S_x (S_y) is the pixel value in image Sobel-X (Sobel-Y), then:

$$|M| = \sqrt{S_x^2 + S_y^2} \text{ and } \Theta = \arctan\left(\frac{S_x}{S_y}\right)$$

We used bit-shifts for implementing most of the multiplications and divisions from the original code, a solution that limited the number of expensive operations and allowed for higher vectorization of the code.

5.4. Texture

Texture refers to a visual pattern or spatial arrangement of the pixels in an image. In MARVEL, texture features are derived from the pattern of spatial-frequency energy across image subbands. We recursively applied a quadrature mirror filter (QMF) bank on the low frequency to decompose an image into low-pass and high-pass spatial-frequency bands in a wavelet fashion. In this study, the Johnston filter QMF12a was used in the filter bank. The separable QMF

filter can further reduce the computational complexity of the QMF wavelet decomposition. A texture vector was obtained by computing the variances of the high frequency outputs of the wavelet filter bank in each iteration. We computed our texture vector of size 12 from four iterations of the QMF wavelet decomposition.

Texture feature extraction involves color conversion, 1D QMF decomposition in both the vertical direction and the horizontal direction, and variance computations. The part of the image to be processed by an SPE is DMA transferred from the main memory to its local store in a sliced fashion (with the slice size equal to the length of the filter bank). Filtering first has to be performed on every column of the image slice and then it has to be performed on every row to get 4 sub-band images (LL, LH, HL, HH). As a 128-bit register can accommodate four single precision floating point numbers, the QMF filtering in the vertical direction was conducted in a 4x4 block-based fashion in order to fully utilize the SIMD feature of the SPE. With the characteristics of the selected filter bank and SIMD operation features, we were able to perform QMF decomposition using a fast implementation which derives the outputs of the high frequency QMF filtering from its low frequency outputs without multiplications. After every iteration, the variances of each of the LH, HL, and HH subband images are computed again as part of the texture vector.

5.5. Concept Detection via SVM Testing

The testing phase for SVM models has, at its heart, a dot-product of fixed-length vectors. A feature vector for an image is processed against a collection of vectors in the model. If a model contains 40 such vectors, then the numeric results of each dot-product are combined using an appropriate weighted combination function. The dot product is implemented using SIMD operations on contiguous floating-point data.

6. PERFORMANCE

To test the performance of the application, we have used a reduced MARVEL main loop, and run it against a test base of one hundred 352x240 pixel RGB images, all of them stored in JPEG format. For each image, the application performs the following steps:

1. Read the file, decompress, and generate the RGB components of the image.
2. Extract image features (color histogram, color correlogram, texture, and edge histogram).
3. Perform concept detection for each feature.

The reported times have been computed as an average over all the test images classified against one model.

We measured the application performance on (a) an Intel Centrino at 3.00 GHz with 1.0GB RAM and (b) a 3.2GHz Cell blade with 512MB RAM and 2 Cell processors (we only used one). For simplicity and comparison fairness, each

SPE algorithm was run on a single SPE. The single-SPE performance numbers are provided below. With eight SPEs on a chip, and since images represent independent work, it should be possible to scale these results with the number of processors; this parallelization is left to future work.

Table 1. Performance comparison

Processor	CH [ms]	CC [ms]	TX [ms]	EH [ms]	SVM [ms]	Total work
Intel	24.65	131.78	16.18	76.48	1.99	251.08
PPE	48.07	321.87	32.59	169.58	4.55	576.65
1 SPE	0.89	6.16	2.04	2.57	0.41	12.07
Speed-up	27.70	21.39	7.93	29.76	4.85	20.8

As shown in Table 1, the performance improvements for each component algorithm running on a single SPE are very impressive. However, due to the different contributions of these kernels in the overall application execution time, the overall speed-up factor is likely to be lower.

For testing the performance of the complete application, we have statically mapped one kernel per SPE (thus, using 5 SPEs), and we have run two scenarios: (1) an SPE sequential scenario, close to the original MARVEL code, where the kernels are executed sequentially, and (2) an SPE parallel scenario, where the feature extractions are run in parallel. For the sequential execution, we have obtained a speed-up factor of 13, while for the parallel version, the overall speed-up factor was 18.8. The differences to the theoretical speed-up are due to the overhead induced by running the complete application, namely file input/output operations for both images and SVM models. Note that in both these scenarios, the SPEs are not fully utilized, due to kernels load imbalance. For future work, we plan to evaluate more complex parallelized versions of MARVEL on Cell.

7. CONCLUSIONS

We have described our experience porting the MARVEL framework to the CBE architecture. The performance improvements were significant, and were obtained with modest porting effort.

8. REFERENCES

- [1] "MARVEL: Multimedia Analysis and Retrieval System," *IBM White Paper*. Available at <http://www.research.ibm.com/marvel/>.
- [2] A. Amir, W. Hsu, G. Iyengar, C.-Y. Lin, M. Naphade, A. Natsev, C. Neti, H. J. Nock, J. R. Smith, B. L. Tseng, Y. Wu, and D. Zhang, "IBM Research TRECVID-2003 System," *Proc. NIST Text Retrieval Conf. (TREC)*, Gaithersburg, MD, Nov. 2003.
- [3] A. Natsev, M. Naphade, and J. R. Smith, "Semantic Space Processing of Multimedia Content", *Proc. ACM SIGKDD*, Seattle, WA, Aug. 2004.
- [4] A. Natsev, M. Naphade, and J. Tesic, "Learning the Semantics of Multimedia Queries and Concepts from a Small Number of Examples," *ACM Multimedia (ACM MM)*, Singapore, Nov. 2005
- [5] J. Hennessey, D. Patterson, "Computer Architecture: A Quantitative Approach", 4th edition, Morgan Kaufmann, 2006.