



Intel[®] Technology Journal

Multi-Core Software

Accelerating Video Feature Extractions in CBVIR on Multi-Core Systems

Accelerating Video Feature Extractions in CBVIR on Multi-core Systems

Yurong Chen, Corporation Technology Group, Intel Corporation
Eric Li, Corporation Technology Group, Intel Corporation
Jianguo Li, Corporation Technology Group, Intel Corporation
Yimin Zhang, Corporation Technology Group, Intel Corporation

Index words: contend-based video information retrieval, multi-core, optimization, parallel computing, performance analysis

ABSTRACT

With the explosive increase in video data, automatic video management (search/retrieval) is becoming a mass market application, and Content-Based Video Information Retrieval (CBVIR) is one of the best solutions. Most CBVIR systems are based on low-level feature extractions guided by the MPEG-7 standard for high-level semantic concept indexing. It is well known that CBVIR is a very compute-intensive task, and the low-level visual feature extractions are the most time-consuming components in CBVIR. Nowadays, with the multi-core processor becoming mainstream, CBVIR can be accelerated by fully utilizing the computing power of available multi-core processors.

In this paper, we optimize and parallelize a set of typical visual feature extraction applications in CBVIR. The underlying optimization and parallel techniques are representative of those used in video-analysis applications and can be further used in other applications to maximally improve their performance on multi-core systems. We conduct a detailed performance analysis of these parallel applications on a dual-socket, quad-core system. The analysis helps us identify possible causes of bottlenecks, and we suggest avenues for scalability improvement to make those applications more powerful in real-time performance.

INTRODUCTION

Nowadays, with advances in video capture and storage techniques, the sheer amount of video data has exploded not only in enterprises but also in our homes. Concomitantly, there is an increasing demand for a system that can help end users to index massive amounts of video data for further search, browse, and

management tasks. Digital home-usage media centers are coming into being for this very purpose. Most of these centers consist of two key ingredients: the Content-Based Video Information Retrieval (CBVIR) module and the computing platform.

CBVIR is a computational technique to index unstructured video information in terms of low-level audio/visual features [1]. MPEG-7 is an experimental standard acting as a guideline for low-level audio/visual feature extractions [2]. It includes a set of visual color, texture, shape, and motion descriptors. Since low-level visual feature extraction is the most time-consuming part in CBVIR applications, these applications are much more compute intensive than traditional video decoding/encoding applications. Although typically the indexing can be done in off-line mode, there are many more emerging scenarios that require a real-time or even super-real-time processing capability in a CBVIR system. With the boom in multi-core processors, we can take full advantage of the computing power of today's multi-core platform to accelerate the use of CBVIR applications [3].

In this paper, we optimize and parallelize a set of typical feature extraction applications on a multi-core system. Our results show most of them are much slower than real-time in their original implementations. After serial optimization, however, they become 3.3x faster, and only five of them are still slower than real-time. After the tailored parallelization, the six most compute-intensive applications obtain up to a 7.6x speedup on a dual-socket, quad-core system, which enables them to achieve super-real-time performance.

This paper is organized as follows. First, we briefly review several low-level visual descriptors under the

guidelines of the MPEG-7 experimental standard. Next, we present our optimization and parallelization methodology for low-level visual feature extractions. Then, we show the performance analysis results of the typical feature extraction workloads.

CBVIR AND LOW-LEVEL VISUAL DESCRIPTORS

Video information differs from conventional text or numerical data in that video data require a large amount of memory and special processing operations. Video retrieval is based on how the contents of a sequence of images can be represented. Computational techniques that pursue the goals of indexing the unstructured visual information are called CBVIR [1, 4]. Generally, a typical CBVIR system includes two ingredients: the back-end for video indexing and the front-end for retrieval query processing. The back-end extracts low-level audio/visual features for video data indexing, while the front-end is a query engine that returns retrieval results based on the similarity between query example and indexed video data [4]. A typical system framework is illustrated in Figure 1.

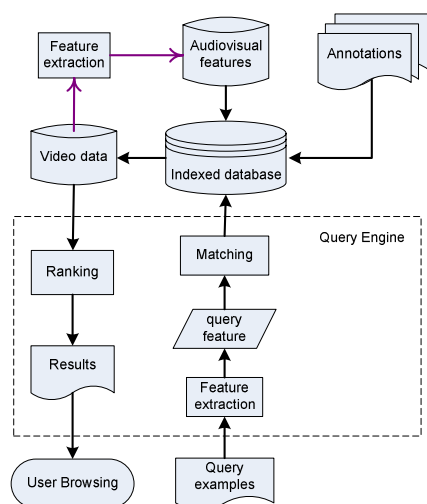


Figure 1: Framework of a typical CBVIR system

The well-known maxim “Garbage in, garbage out” means that good features will greatly improve the retrieval performance of a CBVIR system. Based on this point, the MPEG-7 standard, formally known as the “Multimedia Content Description Interface,” is proposed to guide content retrieval and feature extraction from video data. It includes a set of low-level color descriptors, texture descriptors, shape descriptors, and motion descriptors [2]. Since MPEG-7 is an experimental standard currently, the descriptors are only at the conceptual level. Therefore, in practice, most CBVIR systems just use MPEG-7 as a guideline for

low-level feature extractions [1, 5]. In our experiments, we also use MPEG-7 as a guideline, and we briefly introduce the most-widely used visual features. In each category, we also select one or two typical features with detailed descriptions. These features are widely adopted and have very good retrieval performance [6].

Color Descriptors

Because of its expressive power, color is one of the first attributes used in image description, similarity, and retrieval tasks [7]. MPEG-7 divides color descriptors into several sub-categories: scalable color, color structure, color layout and so on [2]. In practice, there are four widely used color descriptors: Color Histogram, Color Moments, Color Coherence Vector (CCV), and Color Correlogram. The first two can be viewed as scale color descriptors, and the latter two can be viewed as structure color descriptors. In color histograms, overall color distribution can be captured in terms of histogram or low-order moments, but color histograms do not capture any spatial relationships among colors. The CCV is an extension of color histograms, in that it partitions pixels falling in each color histogram bin into coherent pixels and non-coherent pixels.

Color Correlogram is proposed to characterize how the spatial correlation of pairs of colors is changing with the distance [8]. It provides much better performance than color histograms, color moments, and the CCV [6, 8] and has been widely used in CBVIR systems [1, 5]. Color Correlogram extends the co-occurrence matrix method in texture analysis to the color domain. In short, a correlogram is a squared table where the entry at (i, j) specifies the probability of finding a pixel of color c_j at a fixed distance from a given pixel of color c_i . To catch more local spatial information, the co-occurrence can also be defined by banded neighborhoods: this leads to the banded color correlogram. In practice, $\{0, 1, 3, 5, 7\}$ are the most popularly used banded distances.

Texture Descriptors

The textural features describe local arrangements of image signals in the spatial domain or the frequency domain by some spectral transforms. There are many kinds of texture features, such as the Gray-Level Co-occurrence Matrix (GLCM), edge histogram features, multi-resolution simultaneous autoregressive models (MRSAR), wavelet coefficients, and Gabor textures. Specifically, the GLCM is the sufficient statistics of Markov random fields with multiple pairwise pixel interactions. The Edge histogram feature is used to characterize non-homogeneous texture regions. The MRSAR is a random field texture model that characterizes the geometric structure and the

quantitative strength of interactions among neighbors. At present, most promising features for texture retrieval are multi-resolution features obtained from orthogonal wavelet transforms or from Gabor transforms in the frequency domain [7].

MPEG-7 has three texture descriptors: homogeneous texture, texture browsing, and edge histograms. The first two are based on the Gabor transform [2]. The Gabor transform offers the best simultaneous localization of spatial and frequency information. It emerges as an important visual primitive, and it is widely applied in tasks like edge detection, invariant object recognition, and compression [9, 10]. The 2-dimensional (2D) Gabor filters are defined as a series of multi-scale and multi-orientation cosine modulated Gaussian kernels. The Gabor texture representation of images is derived by convolving the image with the Gabor filters and implementing the convolved image efficiently by using Fast Fourier Transform (FFT). The MPEG-7 standard suggests using 6-orientation and 5-scale Gabor filters for the homogeneous texture descriptor and the texture browsing descriptor, which yields one forward 2D FFT for the image and 30 inverse 2D FFTs for the frequency-domain results.

MRSAR is another texture feature studied in this paper, that models the texture as second-order, non-causal Markov random fields [15]. MRSAR uses a 21x21 window sliding across the input image with fixed pixel steps (seven pixels in our experiments) in three resolutions. The least squares estimations are carried out at each resolution independently. Together with the standard deviation of the error term, five parameters are estimated for each resolution and concatenated for a 15-dimensional feature vector. The final feature is the mean and covariance matrix of the 15-dimensional feature on all sliding windows.

Shape Descriptors

The object's shape plays a critical role in searching for similar image objects (e.g., texts or trademarks in binary images or specific boundaries of target objects in images, etc.). In image/video retrieval, one expects that the shape description is invariant to scaling, rotation, and translation of the object. Shape features are less developed than their color and texture counterparts because of the inherent complexity of representing shapes. MPEG-7 supports region-based and contour-based shape descriptors [2]. However, these kinds of shape descriptors rely on the shape quality of shape extraction processes.

Recently, shape context has been proposed as a global shape descriptor, and it has demonstrated great success in image matching, recognition, and retrieval [11, 12]. It

contains two steps: shape extraction and feature formulation. In practice, the shape can be provided by boundary detector, edge detector, or segmentation boundary. Our implementation adopts the simplest Canny edge detector. For each shape point p , it calculates the distance r and orientation θ between the point p and other shape points, and then it quantizes the pair (r, θ) into nine bins of a log-polar coordinate as shown in Figure 2. The 9-bin histogram is used to represent features at point p . Finally, the histogram of each selected key point is flattened and concatenated to form the context description of the shape.

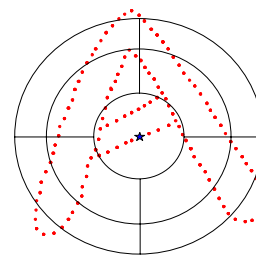


Figure 2: An example of shape context for the reference point

Localization Descriptors

Local descriptors for regions of interest have proved to be very successful in applications such as object recognition, image/video retrieval, and matching different views of object and scene [12]. They are distinctive, robust to occlusion, and do not require segmentation. The idea is to detect image regions that are covariant to a class of transformations, and these regions are then used as support regions to compute invariant descriptors. MPEG-7 contains a region locator and spatial-temporal locators [2]. In this paper we only discuss one of the most widely used localization descriptors: the scale-invariant feature transform (SIFT), which is a known invariant to changes in illumination, image noise, scaling, and small changes in viewpoint [13].

SIFT feature detection can be divided into four steps. The first step detects local extrema in scale-space. SIFT progressively blurs the input image with the Gaussian kernel, resulting in a series of blurred images. Then, each blurred image is subtracted from its direct neighbors (called scale space) to produce a new series of difference of Gaussian (DoG) images. Thereafter, a specific blob detection is conducted at each pixel in the image by comparing the pixel to its eight direct neighbor pixels and 18 neighbor pixels from direct neighbored blur images in the scale space. The second step localizes key points from the extrema in scale space by removing some lower-contrast and noise points. The third step assigns orientation for each key point, and

computes histograms of gradient directions in a 16x16 window at each key point. The fourth step formulates the key point descriptor, which is a 128-dimensional vector of the normalized histogram.

Motion Descriptors

There are four motion descriptors: camera motion, motion trajectory, parametric motion, and motion activity in MPEG-7, which characterize 3-D camera motion parameters, temporal evolution of key points, the motion of regions, and the intensity or pace of motion, respectively [2]. Some MPEG video compression methods already encode macro-block level motion vectors. However, when the pixel-level or object-level motion estimation is required, we must resort to other techniques such as optical flow.

As motion can be represented as vectors originating or terminating at pixels in a digital image sequence, optical flow denotes a vector field defined across the image plane that can wrap images from previous to the next [14]. Estimating the optical flow is very useful in pattern recognition, computer vision, and other image-processing applications. In this work, we study the Lucas-Kanade method, which is known as the most popular two-frame differential method for optical flow estimation. This method tries to calculate the motion between two image frames that are taken at times t and $t+\delta t$ at every pixel position. As a pixel at location (x, y, t) with intensity $I(x, y, t)$ will have moved by δx , δy , and δt between the two frames, optical flow assumes that parts of the objects are the same at the two time slices, i.e., $I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t)$. With first-order Taylor expansion of the left side, and omitting higher-order terms, we have the basic constraint $I_x V_x + I_y V_y + I_t = 0$. The Lucas-Kanade method assumes that the flow (V_x, V_y) is constant in a small window with n pixels, and then it yields n linear equations when taking the n pixels into the basic constraint. Since there are more equations than unknown variables (i.e., $n > 2$), the system is over-determined and can be solved by the least squares method.

OPTIMIZATION AND PARALLELIZATION METHODOLOGY

In this section, we present an optimization and parallelization methodology, characterize different schemes and issues in parallelization, and provide some insights on how to parallelize these video analysis features on a multi-core processor.

Serial Performance Optimization

Before diving into the parallelization study, we first describe several optimization techniques to improve the application's performance. Some optimization can improve both serial and parallel performance. Following we show some widely used techniques we used in a CBVIR application optimization:

- Generic optimization techniques, like loop optimizations, etc.
- SIMD optimization to leverage the data-level parallelism (DLP) architecture features provided by the modern processor.
- Cache-conscious optimization to improve data locality. This is more pronounced for the parallel program due to a reduction of last-level cache misses as well as off-chip bandwidth demands.

Besides manual code optimization, we also extensively use Intel® performance libraries to improve performance. The libraries include the Intel® Performance Primitives (IPP) [16] and the Intel® Math Kernel Library (MKL) [17]. For example, Gabor features use the function *fftwf_execute* to execute discrete Fourier transform for Gabor filters. To achieve better performance we modify the linked library from the open sourced FFTW library to the Intel MKL. The FFTs in the MKL are highly optimized for the latest Intel dual-core and quad-core processors and can provide significant performance gains over alternative libraries for medium and large transform sizes.

Parallelism and Parallel Scheme Study

Usually, thread-level parallelism can be exploited in different ways. There are two primary decomposition methods in the design of a parallel program, i.e., data decomposition and task decomposition methods. The former divides the computations among multiple threads based on the different sections of data. The latter operates on a set of tasks that can run in parallel. Both types of parallelism can be used in the same program and no one method is always better than the other. However, in a CBVIR system, the majority of the work is conducted on 2-D images, which have abundant data parallelism at the picture-level, row-level, and even pixel-level. The selection of data parallelism is a natural choice to make use of the inherent parallelism. Further, to meet the real-time processing capability for these on-line video applications, it is important to extract the fine-grained parallelism within each image instead of exploiting the coarse-grained parallelism at the frame level.

We perform a detailed analysis on these visual features, and we restructure the data and code in order to facilitate the use of threading models. In the following section, we use several examples to demonstrate how to design proper parallel schemes for CBVIR applications.

The major work of the color correlogram consists of counting the color histograms for each pixel. The most straightforward way to do this is to partition the image into several tiles and have each tile accumulate the color data individually. To mitigate the lock contention overhead, in contrast to maintaining only a single histogram buffer, each thread is allocated a local histogram buffer to store the counting data individually. At the end of the parallel region, a reduction operation is conducted to accumulate the private data in each thread. To replicate the thread-private local buffers, we need an additional 20KB of memory per thread. In this way, we reduce the synchronization overhead and achieve better scalability performance. The pseudo code is shown in Figure 3:

```
malloc_local_histogram_array();
#pragma omp parallel
{
    #pragma omp for schedule(dynamic) nowait
    for(int y=0; y<height; y++)
        for(int x=0; x<width; x++)
            calc_correlogram(y, x);
}
merge_result_to_global_histogram_array();
free_local_histogram_array();
```

Figure 3: The color correlogram code example

As shown in Figure 4, the parallelization of Gabor can be conducted at different granularities, such as filter level and FFT level. As we need to convolute images with multiple filters and transform them into frequency domains, the most straightforward way to do this is to perform coarse-grained-level parallelization on these independent filters. The filter-level parallelization scheme can make full use of the underlying processing capabilities with minimal effort. However, it has to prepare local buffers and construct an FFT plan for each filter. This leads to much larger memory consumption, however, and its working set cannot fit well into the last-level cache on a multi-core processor. In addition, the parallelism is also limited by the number of filters, e.g., in Gabor we only have 30 (5x6) filters: when the thread number goes beyond 16, it will incur significant load imbalance. Therefore, exploiting fine-grained parallelism within each filter is also equally important to better express the inherent parallelism. There are three kernels: convolution process, inverse FFT transform, and magnitude computing within each filter iteration.

They can all be parallelized in a fine-grained fashion. As depicted in Figure 4, the parallelization of the convolution and magnitude computing kernels is straightforward. The FFT procedure is also internally parallelized by the MKL. The FFT-level parallelization holds a smaller working set by maintaining only one data copy for all the filters, which greatly improves the cache locality performance and reduces the off-chip memory bandwidth requirements. However, it suffers from fine-grained parallelization overhead and some non-parallel regions, e.g., the preparation stage in the FFT kernel. These will hurt the overall scaling performance. Therefore, we make a detailed comparison between these two parallel schemes, and we choose the one which has the best scaling performance in the final experiments.

```
Filter level parallelization
#pragma omp parallel for dynamic
for(int i=0; i<filter_number; i++)
{
    for(int k=0; k<image_size; k++)
        convolution(i,k);

    fftwf_execute(inverse_FFT_plans[i]);
    for(int k=0; k<image_size; k++)
        compute_magnitude(i,k);
}

FFT level parallelization
for(int i=0; i<filter_number; i++)
{
    #pragma omp parallel for schedule(static)
    for(int k=0; k<image_size; k++)
        convolution(k);

    // fftwf_execute is parallelized in the Intel® MKL
    fftwf_execute(inverse_FFT_plan);

    #pragma omp parallel for schedule(static)
    for(int k=0; k<image_size; k++)
        compute_magnitude(k);
}
```

Figure 4: Gabor parallelism selection

When designing a proper parallel scheme for an application, the best parallel scheme may not come from the best optimized serial algorithm. OpticalFlow (LK method [14], in OpenCV), uses a round-robin buffer to store seven rows of image data and traverses the image in a top-down manner. It has a very good data locality performance. However, the effort for parallelization is not trivial because the data in the buffer written by one thread will be used by another thread. We, therefore,

need to use locks to protect this buffer. Frequent use of locks deteriorates the scaling performance. To break the data dependency among the threads, we use the original algorithm without employing an intermediate buffer for serial program acceleration. It turns out that we achieve much better scaling performance by simply performing parallelization at the row level.

To summarize, to obtain the desired parallel performance, the optimum parallelism depends on the decomposition method used and whether it resulted in the highest scalability performance, and it also depends on the data manipulation techniques and whether they efficiently improve the cache and memory utilizations. In addition, when a serial algorithm is not easy to parallelize in a straightforward way, we may resort to other ways to change the data structure and the control flow, or even modify the algorithm to make it more amenable to parallelization.

Parallel Performance Optimization

After studying the parallelism in the CBVIR applications, we further enhance their performance on multi-core processors. We use several Intel[®] software tools to analyze the parallel programs. For instance, we specify parallelism using OpenMP directives and compile using the Intel compilers. We use the Intel[®] Thread Checker [18] to test the correctness of the program, and the Intel[®] Thread Profiler [18] to collect parallel metrics for bottleneck identification. Furthermore, to understand the cache behavior, we use the Intel VTune[™] Performance Analyzer [19] to collect different levels of cache data.

In parallelizing the CBVIR applications, we identify the parallel bottlenecks and classify them into three categories:

- **Load imbalance.** Load imbalance in a parallel section is a function of the variability of the size of the tasks and the number of tasks. For moderate multi-core processors, it is essential to keep all the cores busy by load balancing the tasks and minimizing overhead. If one core spends more time than the other cores working, the unbalanced load becomes a limiting factor for performance. In CBVIR, we use several techniques to improve the load balance performance, e.g., in MRSAR, a 2-dimension loop is merged into one dimension to enlarge the independent tasks. For almost all the workloads, we use a dynamic scheduling policy to minimize the load imbalance. Particularly, in SIFT, we manually use a “guided” scheduling policy, and the task size is chosen depending on the tasks within each parallelization loop. Since the tasks vary greatly in each iteration when the image in

SIFT is downscaled, a guided scheduling policy helps to balance the size of tasks and scheduling overhead.

- **Synchronization overhead.** Often threads are not totally independent, which forces the program to add synchronization to guarantee the execution order of the threads. The frequent synchronization calls and the associated waiting operations will degrade the scaling performance on multi-core processors. Generally the synchronization is present in the form of critical section, lock, and barrier in the OpenMP implementation. In CBVIR, we largely eliminate the locks by carefully selecting the proper parallelism, e.g., we design a lock-free mechanism in the color correlogram to reduce the synchronization overhead. The shared histogram buffer is replicated into several private data copies. Each thread operates on each local data copy non-exclusively to avoid the mutual access of the shared histogram buffer. In addition, we make careful use of buffer manipulation for each thread, since frequent memory allocation/deallocation operations will cause severe lock contentions in the heap, and these requests are essentially run in serial in a parallel region.
- **Cache efficiency and memory bandwidth.** Good cache efficiency becomes even more important when using multi-core processors, since the maximum bus bandwidth remains unchanged. All cores collectively share a fixed-memory bandwidth; thus, it is important to design algorithms that are cache-conscious and can efficiently utilize the multi-core processing capability. In CBVIR, we designed the parallel programs with the cache performance in mind. We choose the most favorable granularity in terms of cache performance, where fine-grained threads are more cache-friendly than the coarse-grained ones, because more often they can make full use of data sharing instead of replicating buffers for each thread. In MRSAR, sometimes the data access patterns for a 2-D matrix is in column major rather than in row major. This breaks the spatial data locality when accessing the elements in the next row: the data are no longer contiguous. We manually transpose the 2-D matrix and selectively traverse the data according to the data access pattern. Furthermore, we use cache blocking techniques to improve the temporal data locality. We segment the whole data set into several tiles. This subset of data which can fit in cache is operated on all at once before moving on to the next set. Since the block of data can be processed several times before moving on to the next block,

this can significantly improve the cache locality performance.

Besides these general parallel-performance-limiting factors, we also investigate a few more issues specific to multi-core processing.

False sharing is a common pitfall in shared memory parallel processing. It occurs when two or more cores/processors are updating different bytes of memory that happen to be located on the same cache line. Since multiple cores cannot cache the same line of memory at the same time, when one thread writes to this cache line, the same cache line referenced by the other thread is invalidated. Any new references to data in this cache line by the second thread results in a cache miss and potentially huge memory latencies. Therefore, it is important to make sure that the memory references by the individual threads are to different non-shared cache lines. We manually resolve false sharing issues in two kernels of CBVIR, i.e., Shape Context and SIFT, by padding each thread's data element to ensure that elements owned by different threads all lie on separate cache lines. False sharing problems can also be identified during the tuning stage using the VTune Performance Analyzer, either through looking at some specific performance counters or identifying unexpected sharp increases in last-level cache misses.

Another specific performance-tuning technique is using a thread affinity mechanism to improve the cache performance. This minimizes the thread migration and context switches among cores. It also improves the data locality performance and mitigates the impact of maintaining the cache coherency among the cores/processors. Since multi-core processors are likely to have a non-uniform cache architecture (NUCA), the communication latency between different cores varies depending on its memory hierarchy. We use different thread scheduling policies to address this problem. When we find that a group of threads has high data sharing behavior, we can schedule these threads to the same cluster to utilize the shared cache for data transfer. (A cluster is a collection of closely coupled cores, e.g., two cores sharing the same L2 cache in an Intel[®] Core[™]2 Quad processor is termed a cluster.) On the other hand, for applications with high bandwidth demands, we prefer to schedule the threads on different clusters to utilize aggregated bandwidth. For instance, in SIFT, after the row-based parallelization, the image chunk assigned to one thread/core will be used by the other threads. Significant coherence traffic occurs when the image data do not reside in cores sharing the same last-level cache. Therefore, thread scheduling in the same cluster will mitigate the data transfer between

loosely coupled cores that do not reside in the same cluster.

PERFORMANCE ANALYSIS ON MULTI-CORE SYSTEMS

In this section we first show twelve typical visual feature extraction workloads, which are accelerated by serial optimization. Then we parallelize six of the most compute-intensive workloads with the methodology introduced in the previous section. We evaluate the performance of these workloads on an 8-core system, which is a dual-socket, quad-core machine, with two Intel Core 2 Quad processors running at 2.33GHz. Each socket has four cores, and each core is equipped with a 32KB L1 data cache and a 32KB L1 instruction cache. The two cores on one chip share a 4MB L2 unified cache. The maximum FSB bandwidth is 21GB/s.

For the workloads studied in this work, we carefully choose the data sets to represent realistic scenarios. All the experiments are based on the TRECVID 2005 [20] developing data sets. The 141st and 142nd video sequences are chosen to evaluate the performance, which consists of around one hour of MPEG-1 (352x240 in resolution) videos and 791 key frames. The evaluations are directly performed on the extracted key frames.

Serial Performance Improvement

As shown in Figure 5, more than half of the workloads are formerly slower than real-time, i.e., 30 frames per second (FPS), in the serial performance on an 8-core system. After a series of optimizations, these kernels achieved an average of 3.3x speedup, about 60% of which came from using the Intel highly optimized libraries and the SIMD optimization. Even so, five workloads, Correlogram, MRSAR, Gabor, SIFT, and OpticalFlow, are still slower than real time. To harness the power provided by a multi-core system through exploiting thread-level parallelism, we further parallelized these workloads and analyzed their performance on an 8-core system. In addition, to make our work more comprehensive, we also included a representative shape descriptor, Shape Context, in the parallelization study.

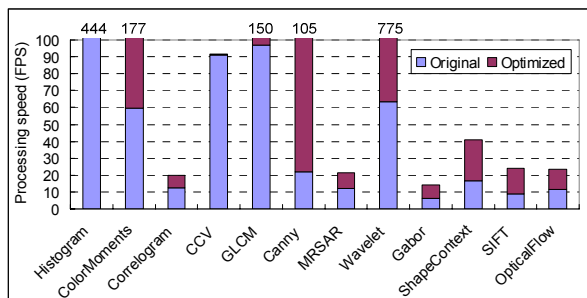


Figure 5: Serial processing speed (FPS) of CBVIR workloads on an 8-core system

Performance Scalability Analysis

These six workloads scale very well as the number of threads increases, as shown in Figure 6. Four of them exhibit almost linear speedups and two achieve quite respectable speedups. That is, CBVIR workloads can efficiently use the computational power provided by multi-core processors.

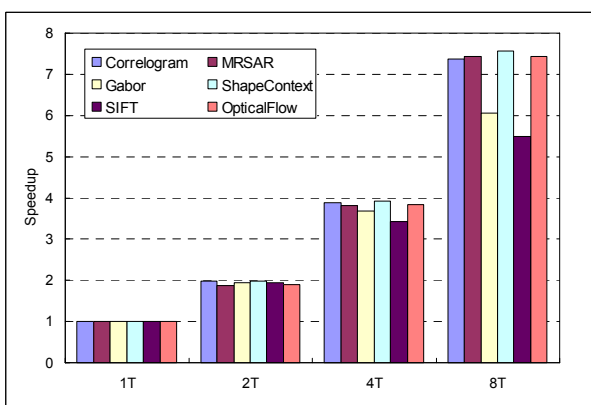


Figure 6: Scalability of parallel CBVIR workloads on an 8-core system

To fully understand the scaling limiting factors on an 8-core system, we characterize the parallel performance from the high-level general parallel overheads, e.g., synchronization penalties, load imbalance, and sequential regions, to the detailed memory hierarchy behavior, e.g., cache miss rates and FSB bandwidth.

We profile them with the Intel Thread Profiler to see their general parallel limiting factors. From Figure 7, we can see that the parallel region dominates in the execution time breakdown, which suggests these CBVIR workloads expose good parallel performance metrics. However, some workloads, especially SIFT, suffer a lot from load imbalance when the number of threads increases to four and eight, which leads to the poor speedup of SIFT. If we assume the parallel region can scale perfectly, Gabor and SIFT should achieve theoretical speedups of 7.6 and 6.2, respectively, on

eight cores. The theoretical speedups are much higher than the practical results shown in Figure 6. Therefore, we believe the scalability of our workloads is also limited by some other factors.

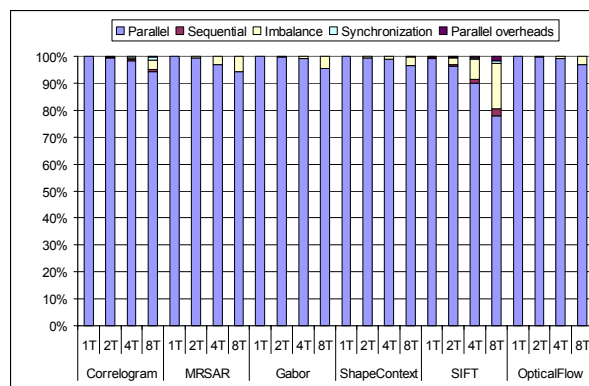


Figure 7: Execution time breakdown

Besides the general scalability performance factors, the memory subsystem also plays an important role in identifying the scaling performance bottlenecks. For further assurance, we get the memory-hierarchy micro-architectural statistics with the Intel VTune Performance Analyzer as shown in Figure 8. The figure shows that L1 cache miss rates vary little with the number of threads, while for some workloads L2 cache performance varies a lot when scaling the thread count. The L2 cache misses for most workloads is reduced when the number of threads increases to four or eight, because the system offers a larger size L2 cache from 4M to 8M and 16M. Since SIFT has a hierarchical parallel decomposition method, the downscale image has to be broadcast to all the private L2 caches after one iteration, thereby incurring significant cache coherency misses when we scale to four and eight cores.

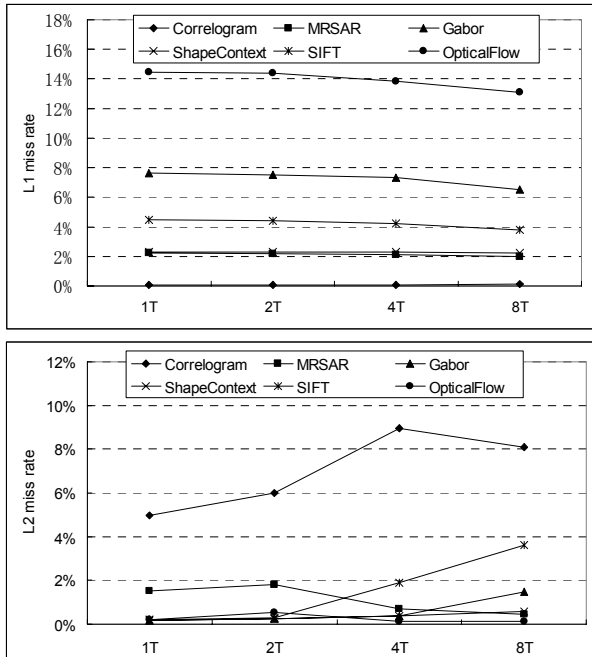


Figure 8: L1/L2 cache miss rates

Generally speaking, memory bandwidth is a key factor that may potentially limit the speedup on multi-core systems. Figure 9 shows how the average FSB bandwidth utilization varies with the number of threads. The bandwidth usages of all workloads are far below the saturated FSB bandwidth capacity supported by the system. This seems to indicate bus bandwidth does not limit the scalability of our workloads on an 8-core system. However, the scalability is limited by the instantaneous bandwidth usage for some workloads, such as Gabor. We perform interval sampling of the memory subsystem behavior over time. Figure 10 shows a representative phase of the bandwidth usage over time for this workload on eight cores. Several modules in this workload have higher bandwidth requirements than the saturated bandwidth provided by the system.

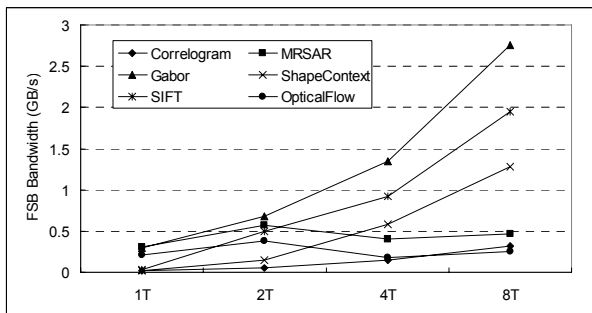


Figure 9: Average FSB bandwidth utilization vs. number of threads

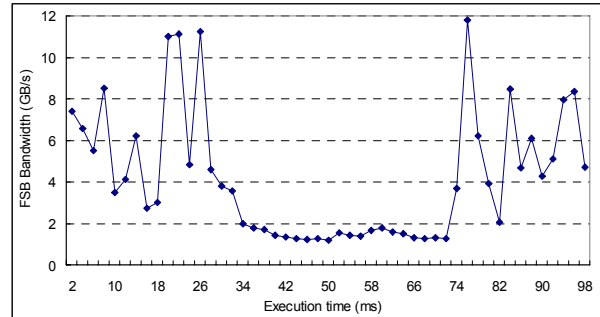


Figure 10: Bandwidth usage over time for eight-threaded Gabor workload

In addition to studying the memory sub-system performance, we also use different thread-scheduling mechanisms to further improve their performance on a multi-core system. As mentioned earlier, there are three scheduling policies: “clustered,” “non-clustered” and “os.” The “clustered” policy tries as much as possible to schedule all the threads to the closely-coupled cores; e.g., it schedules two threads to two cores residing in one chip. In contrast, the “non-clustered” policy tries to schedule the threads to the loosely coupled cores; e.g., it schedules two threads to two cores on two chips instead of one chip. The “os” is the default scheduling policy of the operating system, and it is non-aware of the hardware architecture.

Our results show that some workloads are sensitive to the scheduling policy. Figure 11 shows the scaling performance of Gabor and SIFT using different scheduling policies on an 8-core system. Gabor has better performance with the “non-clustered” policy, while SIFT has better performance with the “clustered” policy. This is because Gabor has a higher bandwidth requirement as shown in Figure 9. The “non-clustered” policy can make full use of the available L2 cache capacity and bandwidth, resulting in better cache performance as depicted in Figure 12. SIFT has better performance with the “clustered” policy because the data can reside in the same L2 cache all the while between several consecutive parallel regions. Otherwise, the data generated by one thread have to be transferred to another core that does not reside in the same L2 cache, yielding significant cache coherency traffic and slowing down the program. As shown in Figure 12, the “clustered” policy in SIFT has far fewer L2 cache misses and a lower FSB bandwidth utilization compared to the “non-clustered” policy. Hence, all the experimental results in the previous sections are obtained by choosing the best policy for each individual workload.

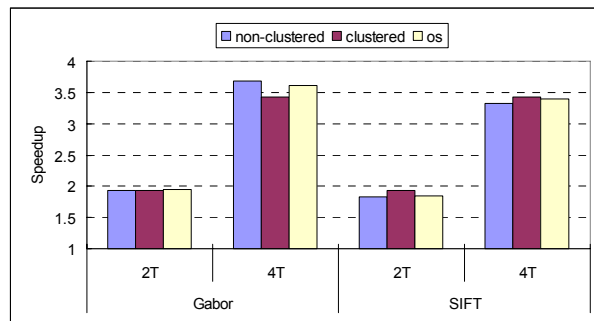


Figure 11: Effects of thread scheduling for two feature extraction workloads on an 8-core system

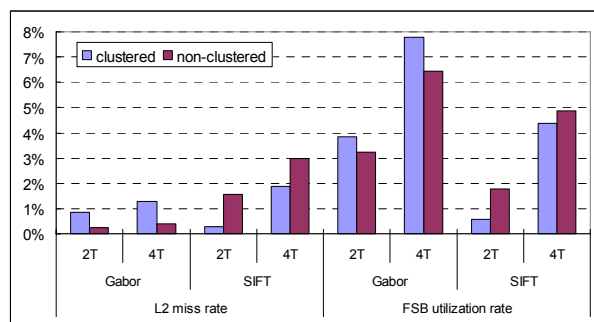


Figure 12: Effects of thread scheduling on L2 miss rate and FSB utilization rate for two feature extraction workloads on an 8-core system

CONCLUSION

CBVIR is becoming one of the best solutions to retrieve useful information from today's massive amount of video data. To accelerate CBVIR on multi-core systems, we optimize and parallelize a set of representative visual feature extraction workloads in CBVIR. We analyze their scalability and memory performance on an 8-core system and draw several conclusions.

Firstly, we choose appropriate parallel schemes for the applications in CBVIR. Exploring different levels of parallelism and choosing the most favorable are necessary to enable optimal performance on multi-core systems. Secondly, we incrementally optimize the parallel performance by mitigating the parallel performance limiting factors, e.g., load imbalance removal, designing cache-friendly data structures, using different thread-scheduling policies, etc. Thirdly, we find most of the CBVIR applications have very good scaling performance. The main scalability limiting factors for SIFT and Gabor are load imbalance and the amount of available system bandwidth. Finally, the CBVIR system is significantly accelerated on multi-core systems and offers enhanced performance to satisfy user requirements.

ACKNOWLEDGMENTS

We acknowledge the encouragement and help that we received from Dr. Bob Liang, Director of the Applications Research Lab. Our thanks go to Prof. Jianming Li and Prof. Bo Zhang from Tsinghua University for collaborating with us on the TRECVID evaluation and for providing some original source code for our analysis. We also acknowledge Qi Zhang for his contribution to this work during his internship with Intel. We also thank the reviewers for their valuable comments.

REFERENCES

- [1] Michael L., Nicu, S., Chabane, D., and Jain, R., "Content-based Multimedia Information Retrieval: State of the Art and Challenges," *ACM Trans. on Multimedia Computing, Communications, and Applications*, 2006, pp. 1–19.
- [2] *MPEG-7 Overview*, ISO/IEC/JTC1/SC29/WG11, N6828, 2004.
- [3] Zhang, Q., Chen, Y., Li, J., and Zhang, Y., "Parallelization and performance analysis of video feature extractions on multi-core based systems," in *Proceedings of the 36th International Conference on Parallel Processing*, 2007.
- [4] Yoshitaka, A. and Ichikawa, T., "A survey on content-based retrieval for multimedia databases," *IEEE Trans. On Knowledge and Data Engineering*, 11(1), 1999, pp. 81–93.
- [5] Smeaton, A. F., Over, P. and Kraaij, W., "Evaluation campaigns and TRECVID," in *Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval (MIR'06)*, pp. 321–330, 2006.
- [6] Ma, W-Y, and Zhang, H-J, "Benchmarking of image features for content-based retrieval," *IEEE Conference on Signals, Systems & Computers*, 1998.
- [7] Manjunath, B. S., Ohm, J.-R., Vasudevan, V., and Yamada, A., "Color and texture descriptors," *IEEE Trans on Circuits and Systems for Video Technology*, 11(6), pp. 703–715, 2001.
- [8] Huang, J., Kumar, S. R., Mitra, M., Zhu, W.J., and Zabih, R., "Spatial Color Indexing and Applications," *International Journal of Computer Vision*, 35(3), pp. 245–268, 1999.
- [9] Lee, T. S., "Image representation using 2D Gabor wavelets," *IEEE Trans. PAMI*, 18(10), pp. 959–971, 1996.

- [10] Manjunath, B. S., and Ma, W-Y, "Texture features for browsing and retrieval of image data," *IEEE Trans. PAMI*, 18 (8), pp. 837–842, 1996.
- [11] Belongie, S., Malik, J., and Puzicha, J., "Shape Matching and Object Recognition Using Shape Contexts," *IEEE Trans. PAMI*, 24(4), pp. 509–522, 2002.
- [12] Mikolajczyk, K. and Schmid, C., "A performance evaluation of local descriptors," *IEEE Trans. PAMI*, 27(10), pp. 1615–1630, 2005.
- [13] Lowe, D. G., "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, 60(2), pp. 91–110, 2004.
- [14] Lucas B. and Kanade T., "An iterative image registration technique with an application to stereo vision," in *Proceedings of Imaging understanding workshop*, 1981.
- [15] Mao, J. and Jain, A.K., "Texture classification and segmentation using multi-resolution simultaneous autoregressive models," *Pattern Recognition*, 25(2), pp. 173–188, 1992.
- [16] "Intel® Integrated Performance Primitives," at <http://www.intel.com/software/products/IPP>
- [17] "Intel® Math Kernel Library," at <http://www.intel.com/software/products/MKL>
- [18] "Intel® Threading Analysis Tools," at <http://www.intel.com/software/products/Threading>
- [19] "Intel VTune™ Performance Analyzer," at <http://www.intel.com/software/products/VTune>
- [20] "NIST, TREC Video Retrieval Evaluation," at <http://www-nlpir.nist.gov/projects/trecvid/>

AUTHORS' BIOGRAPHIES

Yurong Chen is a researcher at the Microprocessor Technology Lab, Beijing. Currently, he conducts research on parallel processing of emerging applications, scalable workloads, benchmarking, and performance analysis for next-generation microprocessors/platforms. He joined Intel in 2004. Before that he did two years' postdoctoral research on large-scale scientific computing in the Institute of Software, Chinese Academy of Sciences. He received his Ph.D. degree from Tsinghua University in 2002. His e-mail is yurong.chen@intel.com.

Eric Li is a researcher in the Microprocessor Technology Lab, Beijing. Currently, he is working on media-mining technology development and performance analysis on multi-core architecture. Prior to this, he was

involved in several projects related to bioinformatics, multimedia, and parallel computing. He received his M.S. degree from Tsinghua University in 2002 and joined Intel that same year. His e-mail is eric.q.li@intel.com.

Jianguo Li is a researcher in the Microprocessor Technology Lab, Beijing. Currently, he works on multimedia mining and parallel algorithm design and implementation. He has been involved in several projects related to sports video analysis and content-based media mining. He received his Ph.D. degree from Tsinghua University in June 2006 and joined Intel after graduation. His e-mail is jianguo.li@intel.com.

Yimin Zhang is a researcher in the Microprocessor Technology Lab, Beijing. He leads a team of researchers working on various statistical computing techniques and their scalability analysis, recently focusing on media mining, data mining, etc. He joined Intel in 2000. At Intel, he has been involved in several projects related to natural language processing and speech recognition, especially focusing on Chinese-named entity extraction and DBN-based speech recognition. He received his B.A. degree from Fudan University in 1993, his M.S. degree from Shanghai Maritime University in 1996, and his Ph.D. degree from Shanghai Jiao Tong University in 1999, all in Computer Science. His e-mail is yimin.zhang@intel.com.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, IPLink, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, VTune, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Intel's trademarks may be used publicly with permission only from Intel. Fair use of Intel's trademarks in advertising and promotion of Intel products requires proper acknowledgement.

*Other names and brands may be claimed as the property of others.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Bluetooth is a trademark owned by its proprietor and used by Intel Corporation under license.

Intel Corporation uses the Palm OS[®] Ready mark under license from Palm, Inc.

Copyright © 2007 Intel Corporation. All rights reserved.

This publication was downloaded from
<http://www.intel.com>.

Additional legal notices at:
<http://www.intel.com/sites/corporate/tradmarx.htm>.

For further information visit:

developer.intel.com/technology/itj/index.htm