

Parallelization and Performance Analysis of Video Feature Extractions on Multi-Core Based Systems

Qi Zhang^{† ‡}, Yurong Chen[‡], Jianguo Li[‡], Yimin Zhang[‡], Yinlong Xu^{† *}

[†] Dept. of Computer Science, Univ. of Science and Technology of China

[‡] Intel China Research Center, Intel Corporation

yurong.chen@intel.com

Abstract

Content-Based Video Information Retrieval (CBVIR) has becoming one of the best solutions for retrieving useful information from today's video information explosion. And with the rapid development of modern technologies, CBVIR is emerging as a mass market desktop application. There is evidence that visual feature extraction is the most time-consuming part in a CBVIR system. In this paper, we implement three video visual feature extractions in parallel by exploring different kinds of thread-level parallelism. We also conduct detailed scalability and memory performance analysis on two multi-core based systems, in order to gain more insights into video-analysis related applications on future multi-core systems. From our analysis we identify the likely causes of bottlenecks in these kinds of applications and suggest ways to improve scalability.

1. Introduction

With the fast increasing of video data, video content analysis becomes more and more important for both personal video data analysis (retrieval, editing, etc), and general video search engine (Google, Yahoo, etc). More specifically, Content-Based Video Information Retrieval (CBVIR) as a common solution for video retrieval attracts more and more attention both in research community and industry. An experimental content based video search standard, MPEG-7 [1] has been proposed by ISO/IEC, which focuses on multimedia content description interface. It describes multimedia content so that users can search, browse, and retrieve the content more efficiently and effectively than with existing mainly text-based search engines. However, some MPEG-7 descriptors, especially visual descriptors, are more complex than the video encoding and decoding.

To promote progress in content-based retrieval from large amounts of digital video, the National Institute of Standards and Technology (NIST) holds an annual workshop on video retrieval evaluation (TRECVID). The participants include not only research groups but also many industry organizations such as IBM, NEC, Microsoft, etc [9]. Most TRECVID systems adopt the MPEG-7 experimental standard to extract different kinds of low-level features for high level concept detection work. However some feature extractions are very time-consuming. Some participants reported that the time consumption of their system was about 600 days on a single processor PC [2]. Nowadays, multi-core systems have become mainstream, which makes it feasible to accelerate the TRECVID task by fully utilizing these available multi-core systems.

In this paper, we optimize and parallelize three most compute intensive low-level visual feature extraction kernels from the Tsinghua University's TRECVID system [2] on two Intel multi-core based systems (an Intel® Conroe desktop PC, and a 4-U Intel® Xeon 7130M server). The underlying optimization techniques and parallel schemes are representative in video-analysis applications, and can be further used in other applications to maximally improve their performance on multi-core systems. We also conduct a detailed performance analysis of the three kernels on these two dual-core based systems. The above efforts help us identify the possible causes of bottlenecks, and we suggest avenues for scalability improvement to make those applications more powerful in real-time performance.

The reminder of this paper is organized as follows. Section 2 provides an introduction to the three visual feature extractions of the TRECVID system. Section 3 describes the optimization techniques and the parallel schemes. The experimental results and application characterizations are reported in Section 4. Finally, we conclude in Section 5.

*This work was partly supported by the NSFC No.60533020.

2. TRECVID Kernel Description

In this paper, low-level feature extractions are selected for our study based on the following criteria: (1) high computational complexity, (2) widely used with good retrieval performance, (3) representative of different optimization scheme. Finally, three kernels are emerged out. They are Gabor texture features [3], Markov Random Filed texture features (specifically, multi-resolution simultaneous autoregressive models, MRSAR) [4], and color correlogram features [5]. These three features are widely used in content-based image/video retrieval systems, and reported very good performance by many researchers [6, 7]. We briefly review their algorithms in this section.

2.1. Gabor Texture Features

Gabor filters offer the best simultaneous localization of spatial and frequency information. It has been widely applied in image processing tasks such as edge detection, invariant object recognition, and compression [8]. Gabor texture feature also has emerged as an important visual primitive for search and browsing [5]. The 2-Dimensional (2D) Gabor filters are defined as follows when assumes $\sigma_x = \sigma_y = \sigma$ [8]:

$$\Psi(z, \sigma_s, \theta_o) = \frac{1}{2\pi\sigma_s^2} \exp\left\{-\frac{\|z\|^2}{2\sigma_s^2}\right\} \left[\exp\left\{\frac{jx'\kappa}{\sigma_s}\right\} - \exp\left\{-\frac{\kappa^2}{2}\right\} \right] \quad (1)$$

$$z = (x', y'),$$

$$\begin{cases} x' = x \cos \theta_o + y \sin \theta_o \\ y' = -x \sin \theta_o + y \cos \theta_o \end{cases}$$

where x, y are pixel position in spatial domain, κ is a parameter for filter bandwidth, θ_o is the filter angle for o -th orientation, and σ_s are the Gaussian deviation for s -th scale, which is proportion to the wavelength of the filters.

The Gabor representation of images is derived by convolving the image with the Gabor filters:

$$G_{s,o}(x, y) = I(x, y) \otimes \Psi(x, y, \sigma_s, \theta_o) \quad (2)$$

where \otimes is the symbol for spatial convolution, and $I(x, y)$ is an input image. This convolution can be fast implemented by Fast Fourier Transformation (FFT):

$$G_{s,o}(I) = IFFT2\{FFT2(I) * FFT2(\Psi_{s,o})\} \quad (3)$$

where $IFFT2$ refers to inverse 2D FFT, and $*$ indicates the production between corresponding elements.

The MPEG-7 experimentation standard suggests Gabor filters based homogeneous texture descriptor. The image is filtered with 6-orientation and 5-scale

Gabor filters, and the means and standard deviations of the filtered outputs in the frequency domain are used as the descriptor.

In our implementation, since the filter parameters are fixed for all input images, the frequency domain filters $FFT2(\Psi_{s,o})$ can be pre-calculated to save some computations, and the Gabor texture feature extraction requires one forward FFT for each input image, $5 \times 6 = 30$ element-based-production (i.e. the $*$ operation in frequency domain in Equation 3) between images and filters, and 30 inverse FFTs.

2.2. MRSAR

MRSAR has been shown one of the best texture feature descriptors by many performance evaluations [3, 6, 7]. The MRSAR method models the texture as a second-order non-causal Markov random fields. MRSAR is carried out in a 21×21 window sliding across the input image with fixed pixel steps (7 pixels in our experiments) on three resolutions. For a given resolution k , the model is defined as

$$g(i, j) = \sum_{(m,n) \in N_k} a_k(m, n) g(i-m, j-n) + n_k(i, j) \quad (4)$$

where N_k is the employed neighborhood of the pixel (x, y) at resolution k , $g(i, j)$ is the gray level values in the image, $a_k(m, n)$ is the model coefficients, and $n_k(i, j)$ is the error term associated with the model. MRSAR assumes the model is symmetric, i.e., $a_k(m, n) = a_k(-n, -m)$. Each pixel in the 21×21 window is characterized by an underlying four parameters autoregressive model at three different resolutions using sub-windows size 5×5 , 7×7 and 9×9 . 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.

2.3. Color Correlogram

The color histograms, moments, and sets do not involve local relationships among the neighborhood pixels. Color correlogram has been recently proposed to characterize how the spatial correlation of pairs of colors is changing with the distance. It has shown to provide much better performance than color histogram, moments etc [5, 6], and been widely used in TRECVID systems [2, 10].

Let I be an $m \times n$ image, for a pixel of $p=(x,y)$, denote the color by $I(p)=c$. Correlogram adopts infinite norm to measure the distance between two pixels, i.e., $|p_1 - p_2| = \max\{|x_1 - x_2|, |y_1 - y_2|\}$. For a given distance $k \in \{1, \dots, d\}$, the color correlogram of image I for the finite color set $\{c_i\}$ is defined as

$$\gamma_{c_i, c_j}^{(k)}(I) = \Pr_{P_1=c_i} \left[P_2 = c_j \mid |P_1 - P_2| = k \right] \quad (5)$$

When assuming $c_i = c_j$ in the definition, we obtain the auto-correlogram $\alpha_c^{(k)}(I) = \gamma_{c,c}^{(k)}(I)$ which captures spatial information between identical colors. To catch more local spatial information, we can calculate correlogram in a banded distance as follows

$$\bar{\gamma}_{c_i, c_j}^{(k)}(I) = \sum_{k'=k_1}^{k_2} \gamma_{c_i, c_j}^{(k')}(I) \quad (6)$$

which is called banded correlogram.

In our experiments, all colors are quantized into 36 distinct ones in the LUV color space in order to reduce the size of the feature set, and $\{1, 3, 5, 7\}$ for the possible banded distance k .

3. Optimization and Parallelization

Recently, most of the video-analysis applications including CBVIR are expected to apply in real-time environments, but it is rather difficult. To optimize the whole video retrieval system, each kernel must be speeded up. Meanwhile, as the future processor is turning to multi-core architecture rather than higher clock speed, we must do more work to improve the performance of our applications and fully utilize the processing capabilities of multi-core processors.

3.1 Serial Optimization

Prior to parallelization, we perform several serial optimizations to improve the performance of the three applications.

In Correlogram, the major operation is accumulating color co-occurrence times at the neighborhood of each pixel. We observe that a pixel x is at distance k away from another pixel y , meanwhile the pixel y is at the same distance from pixel x . A redundancy exists since the contact is calculated twice for each two pixels. We eliminate the redundant process by merging the two statistics into one. This reduces the statistical operations to half without paying.

While examining the profile data collected by the Intel® VTune™ Performance Analyzer [11], we

observe that the hotspot is the computation of infinite norm distance between two pixels. In the original implementation, each pixel is related to a sliding window, which is a square matrix and the center is this pixel. The distances between the central pixel and all other pixels in this window are calculated. Fortunately, we detect that once given a distance, all the pixels can be determined which are at the certain distance away from the central pixel. So we change the loop's variable from the coordinate into the distance, then the computation of infinite norm distance is not required.

In MRSAR, we find that a 2-dimensional array named *sum* needs to be accessed in two orders, sometimes in row order and sometimes in column order. As we know, there will be poor cache performance due to accessing array in column order. So we keep a reverse copy of the array *sum*, and replace the *sum* with the reverse version when we have to access the array in column order. We improve cache reuse by this operation, and the tradeoff is more memory requirement.

In Gabor, we find the hotspot is the function *fftwf_execute* which executes discrete Fourier transform for the Gabor filters. To achieve better performance we modify the linked library from open sourced FFTW library [12] to Intel® Math Kernel Library (Intel® MKL) Version 9.0 [13]. The FFTs in MKL are highly optimized for the newest Intel dual-core and quad-core processors and can provide significant performance gains over alternative libraries for medium and large transform sizes. We also observe that there are abundant floating point operations around the function *fftwf_execute*. Since almost all modern processors have the ability of processing streaming data, we conduct some SIMD (Single-instruction, multiple-data) optimization [14] to accelerate the long element-production between images and filters. We align data structures on 16-byte boundary, and reconstruct the code to vector operation. Then Intrinsic [14] functions are applied to obtain SIMD instructions.

Besides the techniques mentioned above, we apply other optimization techniques, such as, using pre-alloc to reduce operating system expense and achieve good data localization; using of temporary result caching and sub-expression optimization to remove unnecessary computations; using data blocking and loop unrolling to reduce cache misses. With all of these techniques, the aggregated performance improvement is shown in Figure 1. The testing environment is Intel® Core™ 2 6600 CPU, 2GB RAM, Windows Server 2003, Intel® C/C++ Compiler Version 9.1, and the testing data set is 791 key frames of MPG-1 video.

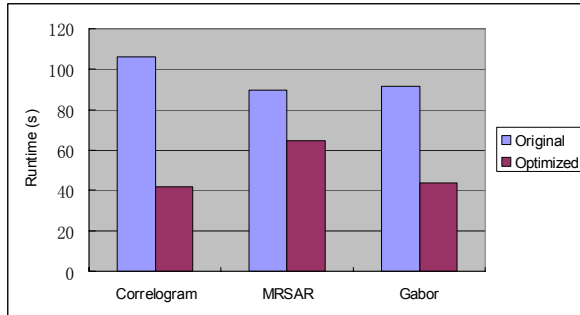


Figure 1. Performance improvement of serial optimization

3.2 Different Parallel Schemes

With the boom of multi-core processor, and the prevalence of shared memory processing, it is important to exploit thread level parallelism within application to fully take advantage of multi-core processing capability. Thread-level parallelism can be exploited in different ways. We could choose processing several images at the same time and we could also choose processing one image in parallel internally. However, in order to implement the real time or on-line processing of these video applications, it is important to extract the fine-grained parallelism in each frame. So in this paper, we focus on how to process each frame as fast as possible with OpenMP [15].

In Correlogram, the major job is counting the color histogram for each pixel. The straightforward processing method of data segmentation can be applied to this condition. We attach the `#pragma omp for` directive to the loop of pixel. Then the OpenMP runtime environment can assign data and task to threads automatically. However, there must be serious contention between those threads, since each thread will access the histogram array. And our tests show as the number of threads increase, the contention grows drastically. So we align independently histogram array memory to each thread and call a data reduction at the end of thread by `#pragma omp critical` directive. This optimization requires only about 20 KB memory for each thread in our experiments. In this way, we reduce synchronization overhead, and achieve better performance of scalability. The pseudo code is shown below:

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

In MRSAR, we try to process several sliding windows in parallel. In the original source code, there is a 2-dimensional loop to scan all the windows. There are only thirty to forty iterations for the first level loop. It will cause some load imbalance when the ratio of iteration number to thread number is very low and the number of iterations is not a multiple of the thread number. So we apply a loop collapsing technique to replace the 2-dimensional loop with a 1-dimensional loop. Thus, there are more iterations which can be assigned to threads, and the runtime of each iteration becomes shorter. In this way, we can significantly decrease the load imbalance between threads.

In Gabor, the parallelization can be conducted with different granularities, such as filter level and FFT level. As we have multiple filters for processing, the most straightforward processing scheme is to perform task-level parallelization (filter level parallelization). Using this scheme, all the filters are assigned to each thread statically. But we have to prepare independently memory and construct FFT plan for each filter. This operation requires a much larger memory consumption to store the input and output of each filter. The pseudo code of this task-level parallelization is shown below:

```
#pragma omp parallel for schedule(static)
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);
    }
}
```

The task-level parallelization scheme can fully utilize the underlying processing capabilities with the least effort. However, it may not perform well on some systems which have small last level cache and low bandwidth of front-side bus (FSB) due to the large number of memory accesses. So exploiting parallelization inside the processing of each filter is necessary. In the data parallelization scheme (FFT level parallelization), we parallelize the convolution process, back FFT transform and magnitude computing separately. The pseudo code of this data parallelization is shown below:

```

for(int i=0; i<filter_number; i++)
{
    #pragma omp parallel for schedule(static)
    for(int k=0; k<image_size; k++){
        convolution(k);
    }
    //the fftwf_execute function has been
    //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);
    }
}
}

```

In contrast to the task-level parallelization scheme, the data parallelization scheme shares memory and FFT plan for all the filters. By checking with Intel® Thread Profiler we find that there is a sequential region in the *fftwf_execute* function which downgrades the scaling performance. We can see that the task-level and data parallelization scheme each has its own advantages and disadvantages. So we implement both the schemes and compare their performance further in the experiments.

In addition, we also parallelize some other loops and choose the scheduling method carefully (static, dynamic or guided). As a result of the above work, we parallelize almost all regions of the three applications.

4. Performance Characterization and Analysis

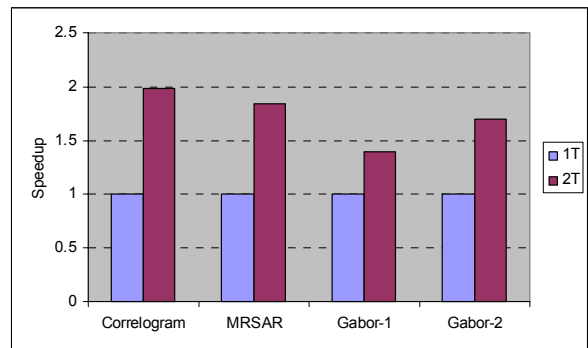
Our experiment and analysis work is based on two Intel multi-core systems. The first is a Intel® Conroe desktop PC mentioned in Section 3.1, and the second is a 4-U Intel® Xeon 7130M server (totally 8 cores) with 8GB shared main memory (see the detailed processor information in Table 1). The configuration of the two systems shows that the 4-U Xeon system has larger last level cache (LLC) capability than the Conroe system.

For software configuration, on both platforms, we use Intel® C/C++ Compiler Version 9.1 to compile the three applications under Windows Server 2003 with full compiler optimization (/O2 /Ot /QxT for the Conroe system, and /O2 /Ot /QxP for the 4-U Xeon system). All the three applications are compiled into 64-bit binaries. Furthermore, we also use the Intel® VTune™ Performance Analyzer and Thread Profiler to collect the data for performance analysis.

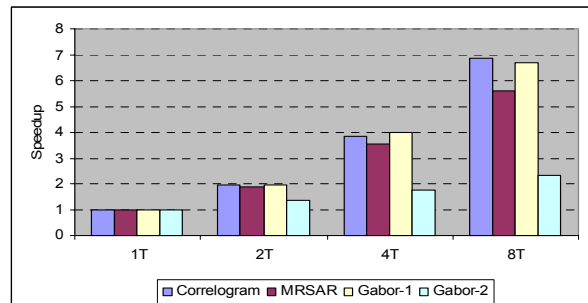
Table 1. CPU information of the two systems

	Conroe	Xeon 7130M
CPU type	Dual-core	Dual-core
Core speed	2.40GHz	3.20GHz
FSB speed	1066MHz	800MHz
L1 data cache	32KB	16KB
L2 cache	4MB (Shared)	2 x 1MB
L3 cache	-	8MB (unified)

All experiments are based on the TRECVID 2005 developing data sets. The 141-th and 142-th video sequences are drawn to evaluate the performance, which adds up to about one hour MPG-1 (352x240 in resolution) videos and contains 791 key frames. Our evaluation is directly running on the extracted key frames.



(a) Speedup performance on the Conroe system



(b) Speedup performance on the 4-U Xeon system

Figure 2. Scalability of Correlogram, MRSAR and Gabor on the two systems

4.1 Scalability Performance Analysis

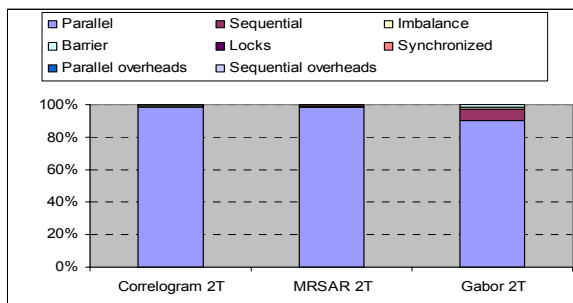
Figure 2 shows the speedup curves of Correlogram, MRSAR and Gabor on the two multi-core systems, respectively. In this figure, Gabor-1 denotes the task-level parallelization implementation for Gabor and Gabor-2 denotes the data parallelization implementation for Gabor. Figure 2(a) shows that Correlogram and MRSAR achieve nearly 2X speedup with 2 threads on the Conroe system. And Gabor-1 gets only 1.4X speedup while Gabor-2 gets 1.7X speedup. Figure 2(b) indicates that the performance of Gabor-2 is very poor on the 4-U Xeon system. All applications except Gabor-2 reach almost linear speedups with 2 to 4 threads on the 4-U Xeon system. As the thread number increases to 8, the speedup of Correlogram and Gabor-1 achieves about 7X, and the speedup of MRSAR is 5.6X.

This experiment shows that for Gabor, the task-level parallelization scheme is suitable on the 4-U Xeon system while the data parallelization scheme is suitable on the Conroe system. In the following experiment, we only analyze task-level parallelization of Gabor on the 4-U Xeon system and data parallelization on the Conroe system.

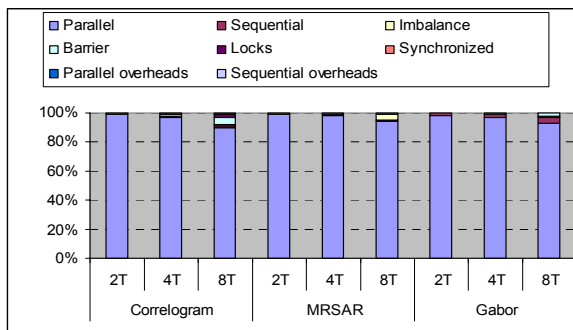
To deeply understand the scaling limiting factors, we characterize the parallel performance from the high level general parallel overheads, e.g., synchronizations penalties, load imbalance, and sequential regions, to the detailed memory hierarchy behavior, e.g., cache miss rates and FSB bandwidth.

We profile them with the Intel® Thread Profiler to see their general parallel limiting factors. From Figure 3, we can see that the parallel region dominates in the execution. On the Conroe system there are very few serial regions in Correlogram and MRSAR. And in Gabor-2 there is 7.4% serial region and 2.3% other parallel overheads. This is the major reason why the speedup with two threads is only 1.7X for Gabor-2 in Figure 2(a). On the 4-U Xeon system, the time percentage of additional expense for parallel (e.g. barrier, locks, and imbalance) grows to 4% for Gabor and 8% for Correlogram with eight threads. Hence, we obtain about 7X speedups with eight threads. Besides this, the most serious parallel limiting factor of MRSAR on the 4-U Xeon system is the load imbalance part. This arises from some loops which are consisted of a few iterations and are not enough to be evenly distributed for all eight threads. However, the percentage of the load imbalance (3.9%) is not so large to lead about 5.6X speedup with 8 threads. As we can see that the aggregate running time of the parallel regions increases from the uni-threaded running to the multi-threaded running. It is highly possible that some

operations run slower in the multi-core configuration than in the single core configuration.



(a) Time percentage on the Conroe system



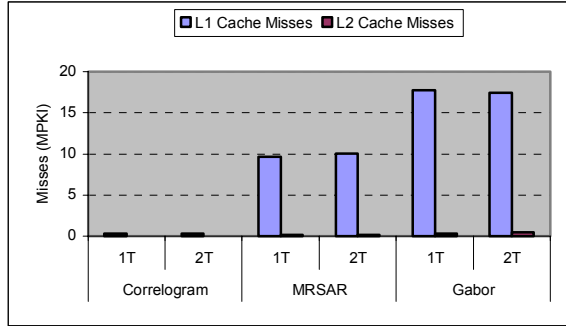
(b) Time percentage on the 4-U Xeon system

Figure 3. Breakdown of time spent in parallel, sequential and other overhead

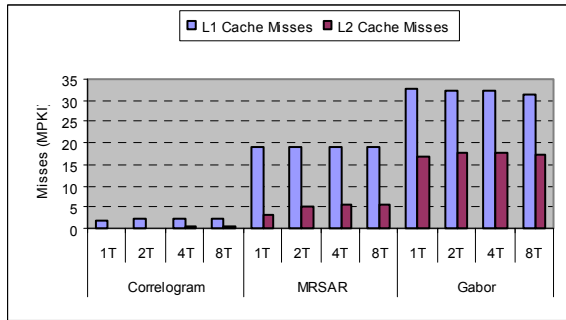
Besides the general scalability performance factors, memory subsystem also plays an important role in identifying the scaling performance bottlenecks. For further assurance, we get the memory-hierarchy micro-architectural statistics with the Intel® VTune™ Performance Analyzer in Figure 4 and Figure 5. Note that we can not get the L3 cache miss data and the bandwidth data of these three kernels on the 4-U Xeon system, since there is no related hardware counters on the target processor.

Obviously the L1 and L2 cache misses on the Conroe system is rather less than on the 4-U Xeon system as shown in Figure 4. This benefit comes from the large L1 cache and the shared L2 cache on the Conroe system. From Figure 4(b), we also observe that on the 4-U Xeon system the cache misses are constant with different thread number for Correlogram and Gabor, but for MRSAR the L2 cache misses grow as the thread number increases. This is because we apply dynamic scheduling for threads to reduce the load imbalance, but this dynamic scheduling destroys the data locality and raise some cache miss. We speculate the L3 cache misses will increase accordingly, since the working set of each thread is larger than 8MB L3

cache. For this reason the average latency to access data is slower in the multi-threaded running than uni-threaded running. And due to the increase of cache misses and load imbalance, the scalability of MRSAR is a little poorer.



(a) Cache misses on the Conroe system



(b) Cache misses on the 4-U Xeon system

Figure 4. Cache misses per kilo instructions (MPKI) of Correlogram, MRSAR and Gabor

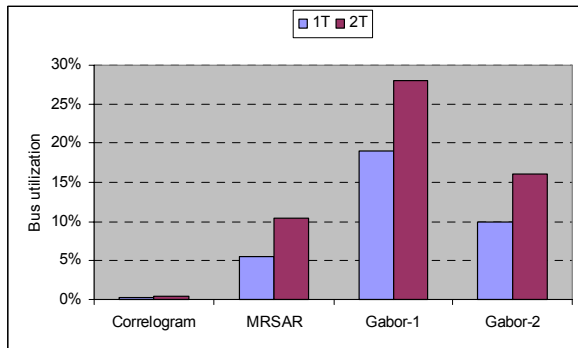


Figure 5. The FSB bandwidth utilization of Correlogram, MRSAR and Gabor on the Conroe system

Generally speaking, memory bandwidth is a key factor which may potentially limit the speedup on multi-core systems. Figure 5 shows the memory bandwidth utilization of Correlogram, MRSAR and

Gabor on the Conroe system. As expected, the FSB utilization rates of the three applications increase with the number of processors and the FSB bandwidth requirement of Gabor is much larger than Correlogram and MRSAR. For Correlogram and MRSAR, the bus remains under-utilized since the largest bandwidth utilization is less than 11%, while for Gabor the utilizations reach 28% with the task-level parallelization and 16% with the data parallelization. This observation shows that the three applications are not bandwidth-limited on the Conroe system, and the bandwidth requirement of Gabor-1 is rather larger than that of Gabor-2. Based on the trend, we can expect the first two applications are also not bandwidth-limited while application Gabor should be a bit bandwidth-limited on the 4-U Xeon system.

4.2 Effects of Thread Scheduling

The dual-core processors in our two target systems both share LLC although they have different memory hierarchy. To maximize the potential of this kind of processors, we study the influence of thread scheduling on the performance of these parallel applications.

In this experiment we use thread affinity [17] to attach one thread to a specific processor, and test the time and speedup performance of the three applications by using three different scheduling policies: “same”, “diff” and “os”. The “same” policy tries to schedule all threads in the same core group as much as possible, e.g. it schedules two threads in two cores on one chip. On the contrary, the “diff” policy tries to schedule all threads on different core groups as much as possible, e.g. it schedules two threads in two core groups on two chips. The “os” is the default scheduling policy of the operation system.

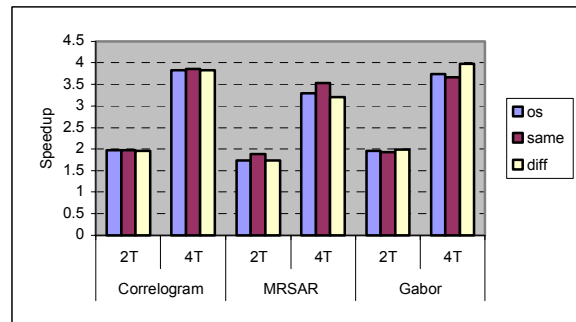


Figure 6. Effects of thread scheduling for the three kernels on the 4-U Xeon system

Figure 6 shows the speedup performance of the three applications using different policies with two and four threads on the 4-U Xeon system. For

Correlogram, the performances of the three policies are almost identical. And for MRSAR, the speedups of “same” are the highest. This is because with the “same” scheduling policy threads can share data well in the unified LLC. For Gabor, the “diff” policy performs the best. This is because the requirement of bandwidth in Gabor is the greatest, and among all these three policies, “diff” can make the fullest use of the available LLC capacity and bandwidth. We also observe that the performance of “os” is right between those of the other two policies. Hence, all experimental results in previous sections are obtained by using “diff” policy for Gabor and by using “same” policy for Correlogram and MRSAR.

5. Conclusion

Visual feature extractions are key techniques for future video-analysis systems. It can help users extract the useful visual information from the explosion of video information they are faced with.

This paper looks at the parallel implementation and runtime performance of three low-level visual feature extraction kernels in the TRECVID system by exploring different kinds of thread-level parallelism. We analyze their scalability and memory performance on two multi-core based systems.

Firstly, to run these applications in parallel on multi-core systems, there are different parallel schemes. This reveals the importance of extracting different kinds of thread-level parallelism for future multi-core equipped SMP systems, which have more complicated memory hierarchies.

Secondly, the main limiters to parallel low-level visual feature extraction are load imbalance and the amount of available system bandwidth. Its scalability performance will increase as the fine-grained parallelism is used and the bandwidth increases.

Thirdly, the scalability performance of this kind of applications is sensitive to the system’s scheduling policies. It is necessary to develop new memory/application-aware scheduling policies for future multi-core systems.

6. References

- [1] J.M. Martinez, “MPEG-7 Overview”, N6828 ISO/IEC/JTC1/SC29/WG11, *Technical report*, October 2004.
- [2] J. Cao, Y. Lan, et al, “Intelligent Multimedia Group of Tsinghua University at TRECVID 2006”, *Proc. TRECVID’06*, 2006.
- [3] B.S. Manjunath, and W.Y. Ma, “Texture features for browsing and retrieval of image data”, *IEEE Trans on PAMI*, 1996, 18 (8): 837-842.
- [4] J. Mao, and A.K. Jain, “Texture classification and segmentation using multi-resolution simultaneous autoregressive models”, *Pattern Recognition*, 1992, 25(2): 173-188.
- [5] J. Huang, S.R. Kumar, M. Mitra, W.J. Zhu, and R. Zabih, “Spatial Color Indexing and Applications”, *IJCV* (35), No. 3, 1999, pp. 245-268.
- [6] W.Y. Ma, and H.J. Zhang, “Benchmarking of image features for content-based retrieval”, *IEEE Conference on Signals, Systems & Computers*, 1998, pp 253-257.
- [7] K. Xu, B. Georgescu, D. Comaniciu, and P. Meer, “Performance analysis in content-based retrieval with textures”, *Proc. 15th International Conference on Pattern Recognition (ICPR’00)*, pp. 4275-4279.
- [8] T.S. Lee, “Image representation using 2D Gabor wavelets”, *IEEE Trans. PAMI*, vol. 18, no. 10, 1996, pp. 959--971.
- [9] W. Kraaij, P. Over, and A.F. Smeaton, “TRECVID 2006 -- An Introduction”, *Proc. TRECVID’06*, 2006.
- [10] M. Campbell, S. Ebadollahi, M. Naphade, et al, “IBM Research TRECVID-2006 Video Retrieval System”, *Proc. TRECVID’06*, 2006.
- [11] Intel® VTune™ Performance Analyzer, <http://www.intel.com/software/products/vtune/>.
- [12] FFTW, <http://www.fftw.org/>.
- [13] Intel® Math Kernel Library, <http://www.intel.com/software/products/mkl/>.
- [14] *Intel® 64 and IA-32 Architectures Optimization Reference Manual*, Intel Corporation, 2006.
- [15] *OpenMP C and C++ Application Program Interface, 2.0 Edition*, The OpenMP Architecture Review Board, Mar, 2002.
- [16] J.B. Huntsman, and R. Rahman, “Improve Performance with Thread Aware Memory Allocators”, *Technical report*, Intel Corporation, 2000.
- [17] J.D. Salehi, J.F. Kurose, and D.F. Towsley, “The Effectiveness of Affinity-Based Scheduling in Multiprocessor Networking”, In *Proc. Of INFOCOMM’96*, 1996.