Parallelization and Characterization of SIFT on Multi-Core Systems

Hao Feng Eric Li Yurong Chen Yimin Zhang
Application Research Lab, Intel China Research Center
{hao.feng eric.q.li yurong.chen yimin.zhang}@intel.com

ABSTRACT

This paper parallelizes and characterizes an important computer vision application — Scale Invariant Feature Transform (SIFT) both on a Symmetric Multiprocessor (SMP) platform and a large scale Chip Multiprocessor (CMP) simulator. SIFT is an approach for extracting distinctive invariant features from images and has been widely applied. In many computer vision problems, a real-time or even super-real-time processing capability of SIFT is required.

To meet the computation demand, we optimize and parallelize SIFT to accelerate its execution on multi-core systems. Our study shows that SIFT can achieve a 9.7x ~ 11x speedup on a 16-core SMP system. Furthermore, Single Instruction Multiple Data (SIMD) and cache-conscious optimization bring another 85% performance gain at most. But it is still three times slower than the real-time requirement for High-Definition Television (HDTV) image. Then we study the performance of SIFT on a 64-core CMP simulator. The results show that for HDTV image, SIFT can achieve an excellent speedup of 52x and run in real-time finally.

Besides the parallelization and optimization work, we also conduct a detailed performance analysis for SIFT on those two platforms. We find that load imbalance significantly limits the scalability and SIFT suffers from intensive burst memory bandwidth requirement on the 16-core SMP system. However, on the 64-core CMP simulator the memory pressure is not high due to the shared last-level cache (LLC) which accommodates tremendous read-write sharing in SIFT. Thus it does not affect the scaling performance. In short, understanding the characterization of SIFT can help identify the program bottlenecks and give us further insights into designing better systems.

1. INTRODUCTION

Image matching is one of the most fundamental topics in the computer vision area and how to generate stable image features is a crucial problem for object matching and feature tracking. Scale Invariant Feature Transform (SIFT) algorithm has been well applied to scene modeling and object recognition since its image features are highly invariant against the translation, rotation and scaling of the image, and are partially invariant to the changes of the lighting and contrast [4]. Because its feature vector descriptors created from image are highly stable and distinctive, image features can be correctly matched with high probability. Though the SIFT algorithm is mature and effective, to generate the feature descriptors is very time-consuming. Moreover, there are many scenarios (e.g. online object recognition) which require SIFT features to be extracted and matched in real-time (30 frames per second - FPS) or even in super-real-time.

In order to take advantage of the computing power of modern multi-core systems to accelerate the SIFT execution, we propose a parallel implementation of the algorithm. We further optimize its performance and present a thorough workload characterization both on a 16-core Symmetric Multiprocessor (SMP) and a 64-core Chip Multiprocessor (CMP) in this paper.

The main contributions of this paper are:

1. We parallelize SIFT on the multi-core systems through exploring its thread-level parallelism and we also present several techniques to further optimize its performance.
2. We evaluate the parallel SIFT performance and conduct a detailed workload characterization of it both on a 16-core SMP machine and a 64-core CMP simulator.

The key findings of this paper are:

1. Thread-level parallelism improves the performance of SIFT by 9.7x ~ 11x on the 16-core SMP machine and 38x ~ 52x on the 64-core CMP simulator. Single Instruction Multiple Data (SIMD) and cache-conscious optimization bring an additional 85% performance gain at most. With those optimizations, SIFT can run in real-time on the 64-core simulator.
2. Load imbalance is the primary factor which limits the scaling performance of SIFT severely. The problem becomes even worse when the thread count increases due to the limited amount of tasks.
3. The workload has high burst memory bandwidth demand on the 16-core SMP system and it affects the scalability of SIFT seriously. Nevertheless, SIFT exhibits moderate memory requirements on the 64-core CMP simulator, since the 16MB shared last-level cache (LLC) significantly reduce the off-chip memory accesses and much higher memory bandwidth is provided by the simulator. Thus, the memory bandwidth demand does not affect the scalability on the 64-core CMP simulator.
4. SIFT exposes tremendous amounts of read-write data sharing because of the true inter-thread communication. But a shared LLC can eliminate the coherence traffics to offer better performance.

The remainder of this paper is organized as follows. Section 2 describes the serial SIFT algorithm. In Sections 3, we present the parallelization and optimization techniques. Section 4 and 5 report the performance result analysis on a

978-1-4244-2778-9/08/$25.00 ©2008 IEEE

16-core SMP platform and a 64-core CMP simulator respectively. Section 6 introduces some related work. Finally, we summarize our study in Section 7.

2. SIFT ALGORITHM

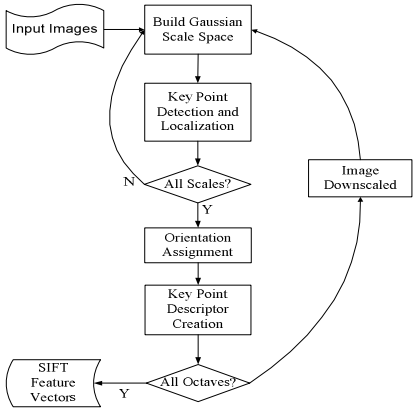


Figure 1 Flowchart of the SIFT Algorithm

In this section, we introduce the serial SIFT algorithm [4]. SIFT is an approach for detecting and extracting local feature descriptors from images. The flowchart of SIFT is shown in Figure 1. In order to generate the set of image features, the algorithm consists of five major computation steps: Building Gaussian Scale Space, Key Point Detection and Localization, Orientation Assignment, Key Point Descriptor as well as Matrix Operations.

Building Gaussian Scale Space (BGSS) Key point candidates for SIFT features are obtained potentially from local extrema of difference-of-Gaussian (DoG) pyramid. As shown on the left of Figure 2, the scale space pyramid (octave) is constructed by convolving the initial image with Gaussian kernels repeatedly. After that, the first image in this octave is downscaled by a factor of two and convolved with the same set of Gaussian kernels to construct the next octave until a certain image size is reached [17]. Finally, as shown on the right of Figure 2, adjacent Gaussian images are subtracted to produce the DoG images.

Key Point Detection and Localization (KDL) The pixels with local maximum or minimum luminance value in the DoG images are considered as key point candidates. Each pixel in the DoG images is compared against its 26 neighbors in the 3x3 region in current and adjacent scales. If the pixel is a local extrema, it is selected as a candidate key point. After removing the low contrast points introduced by noise and edge responses, we obtain the key points finally.

Orientation Assignment (OA) A gradient orientation histogram is computed in the neighborhood of key points in order to determine the key point orientation. The contribution of each neighboring pixel is weighted by the gradient magnitude and a Gaussian window, the size of which is 1.5 times of the key point scale. The peak in the histogram corresponds to the dominant orientation and a key point will be created for that direction as well as any other direction within 80% of the peek value.

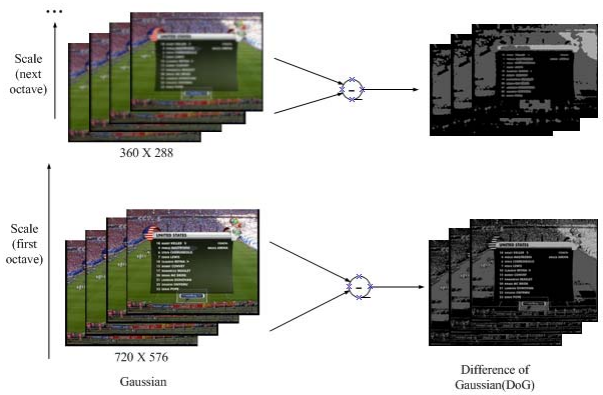


Figure 2 Blurred Images at Different Scales and the Computation of DoG Images

Key Point Descriptor (KD) Once a key point orientation is selected, the feature vector descriptor is computed as a set of orientation histograms in the 16x16 pixel region. Since every 4x4 region is projected onto one histogram and each histogram contains 8 bins for 8 directions, it leads to a SIFT feature vector of 128-element (4*4*8=128) size. The vector is normalized to enhance invariance to changes in illumination [5].

Matrix Operation (MO) After all the key point descriptors in one octave are generated, the Gaussian image is downscaled for the computation of the next octave. The whole algorithm repeats until a specified image size is reached. Some other matrix operations like matrix subtraction are also included in this module.

3. SIFT OPTIMIZATION

In this section, we parallelize SIFT with OpenMP [14] and conduct several optimizations [15] to improve its performance on the multi-core systems. We use the serial algorithm described in Section 2 as the baseline.

3.1 Parallel Algorithm Introduction

There are two main strategies to parallelize the SIFT algorithm, i.e., coarse-grain and fine-grain parallel approaches. As the computations can be operated in different frames independently, the straightforward coarse-grain parallelization is to process different frames in parallel. Though this scheme exploits the highest level of parallelism, it does not scale well because its aggregate working set while processing multiple frames cannot fit into the on-die cache. Thereby, it puts heavy pressure on the memory subsystem and limits the scalability. In addition, this approach only works effectively for off-line video processing. In order to reduce the working set size and provide on-line processing capability, we target to exploit the fine-grain thread-level parallelism within each single frame.

For the fine-grain parallelization, we profiled the serial SIFT algorithm and selected the most expensive modules as the main targets for parallelization. There are four time-consuming modules: BGSS, KDL, OAKD (includes both OA and KD phases described in Section 2) and MO. They make up 99.8% of the total SIFT execution time. As there are abundant row-wise and pixel-wise data parallelisms while processing the images, it is straightforward for each thread to process different chunks of the image data in parallel for BGSS, KDL and MO modules. For OAKD module, since the computation for each key point varies, the key points are dynamically assigned to the working threads during parallel processing.

3.2 Parallel Algorithm Optimization

Besides the parallelization, we also use several optimization techniques to further improve the performance of SIFT on SMP and CMP platforms.

3.2.1 SIMD Optimization

In this work, we dramatically improve the performance of SIFT through SIMD instructions. In order to exploit the data-level parallelism, most modern processors adopt SIMD as an architecture feature such as SSE/SSE-2 of X86 and VMX of PowerPC [9]. The SIMD technology which performs multiple arithmetic or logic operations simultaneously can improve the execution efficiency effectively.

In SIFT most of the computations can be SIMDized through SSE-2, and it achieves a 1.9x speedup on an Intel® Core™ 2 Quad processor. Since BGSS and MO modules are responsible for image convolution and transformation, they are amenable to SIMDization due to their regular data access and manipulation patterns. However, there are still a few factors which limit the SIMD efficiency in SIFT, e.g., different control flow paths, irregular data accesses and atomic operations.

In OAKD module, the 4 elements (pixels) contained in one SSE register may fall into different control flows. In such cases, those pixels have to be processed separately. In order to solve the irregular control flow problem, we use several buffers to record those pixels, and each buffer corresponds to one control path. Then different buffers of pixels can be processed through SSE instructions. However, it incurs significant overhead in buffer manipulation.

In addition, the SIMD efficiency is also affected by the histogram computation[1] for the feature creation in OAKD module, because the unaligned memory accesses and atomic operations prohibit efficient histogram SIMDization. Thus we execute the histogram operations in a serial manner.

3.2.2 Cache/Memory Optimization

Besides the SIMD optimization, we also optimize the cache/memory performance for SIFT. On multi-core platform, good cache efficiency becomes more critical, since the maximum bus bandwidth shared by all cores collectively still remains. Thus, it is important to design algorithms that are cache-conscious and utilize the multi-core processing capability efficiently. In SIFT, we widely apply loop fission and loop interchange to improve cache locality. Meanwhile, we eliminate redundant memory copy operations to save the memory bandwidth usage. For example, the BGSS module employs in-place Gaussian convolution originally. At the end of that phase, it shifts all the pixels inside the convolution window through memory copy operations. We replace that implementation with writing every pixel directly to its final position during Gaussian convolution. It achieves much better performance. Besides, we extensively change the data structure and control flow to eliminate the redundant memory copy operations.

Another cache optimization is to carefully change the data arrangement in order to avoid false sharing in parallel programs. In SIFT we manually add some padding to ensure that the elements owned by different threads lie in separate cache lines.

3.2.3 Thread Affinity Optimization

The thread affinity mechanism [6] can attach one thread to a specific core on SMP or CMP. When a group of threads have heavy data sharing behavior, using the thread affinity to bind them to the cores on the same chip utilizes the shared LLC for fast communication and synchronization. While for the applications with high per-thread bandwidth demand, we schedule the threads to the cores on different chips to utilize the aggregated bandwidth. In the SIFT implementation, we find the affinity mechanism which tries to schedule the threads onto the same socket and even the same chip brings the most benefits. It is because SIFT has a lot of data sharing among threads (will be explained in Section 5). It is beneficial to schedule the threads with lots of data sharing onto the cores sharing the L2 cache. It provides 2% ~ 10% performance gain compared to the default OS scheduling policy. Therefore, we use this scheduling policy throughout the rest of the paper.

4. PERFORMANCE EVALUATION ON THE 16-CORE SMP SYSTEM

In this section, we first evaluate the performance of the parallel SIFT implementation on a real SMP system. The SMP system is a four-socket quad-core server (*HP ProLiant DL580 G5*) with 16 cores totally. Each Intel Core2 Quad processor has four cores running at 1.6 GHz and each core has a 32KB L1 data cache and a 32KB L1 instruction cache. Each of the two pairs of cores on each chip share a 2MB unified L2 cache. Therefore, it leads to a total 16MB L2 cache for the whole 16-core system. The Front Side Bus (FSB) frequency is 1066MHz and the theoretical peak bandwidth is around 33.3GB/s.

The parallel SIFT implementation with OpenMP is compiled by Intel C/C++ Compiler Version 9.1 with maximum optimizations enabled (-O3) under Linux. Some other performance data are collected by Intel VTune performance analyzer [12] and Intel Thread Profiler [11].

[1] Histogram operations are still hard to be parallelized through SIMD. Some special hardware or instructions have been proposed in [7] [16] to solve this problem.

We use two different input data sets in our experiments: MPEG-2 image (size of 720x576, 700 key points) and High-Definition Television (HDTV) image (size of 1920x1080, 1038 key points). To get higher algorithm accuracy, HDTV images are also required, thus we believe both MPEG-2 and HDTV input images are representative.

4.1 Parallel Performance Improvement

The optimization techniques mentioned above bring significant performance gains for SIFT. The performance results on the SMP platform are shown in Figure 3, where "original" represents the SIFT implementation without SIMD and cache optimization. The SIMD speedups are also shown on the graph. From Figure 3, we get following observations:

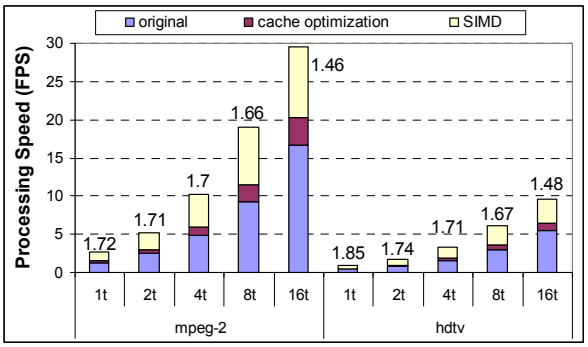


Figure 3 SIFT Performance Improvement

First, on the 16-core SMP system SIFT can marginally meet the real-time requirement for MPEG-2 image. It runs at 30 FPS for 16-thread. But for HDTV, SIFT can run only at 10 FPS that is still far from the real-time performance.

Second, as shown in Figure 3, it is evident that the overall performance of SIFT has been improved dramatically through SIMD optimization which brings a 1.85x speedup for HDTV and a 1.72x speedup for MPEG-2. Considering only BGSS and MO modules obtain the maximum performance gain from SIMD acceleration, KDL module are just partially SIMDized and OAKD module suffers from the irregular control flow and histogram operations, the SSE speedups are reasonable. But as thread number increases, SIMD speedups reduce to only about 1.45x for 16-thread. It is because of the synchronizations and parallelization overhead which cannot be optimized through SIMD.

Third, we find that SIMD offers more performance benefits for HDTV than MPEG-2. In order to understand the reason, we analyze the module runtime breakdown shown in Figure 4. As depicted in that figure, for HDTV image the BGSS and MO modules that are fully optimized through SIMD occupy a much larger proportion of runtime than MPEG-2 image. It accounts for the higher SIMD speedups for HDTV since the BGSS and MO modules have more SIMD benefits than the OAKD and KDL modules.

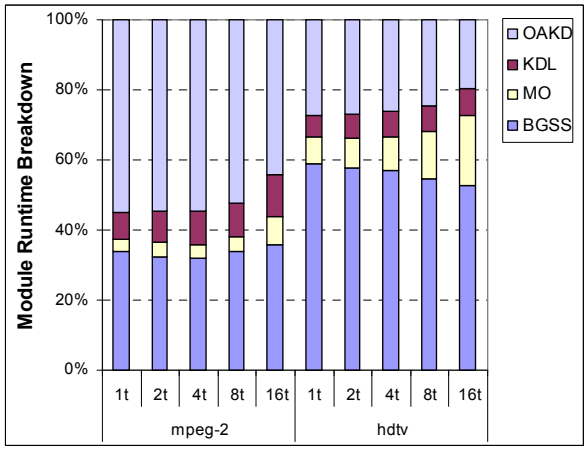


Figure 4 Module Runtime Breakdown

Finally in Figure 3, it is observed the cache-conscious optimization brings an additional 20% ~ 25% performance improvement. Totally, all the optimizations can effectively offer an accumulative 1.8x ~ 2.2x speedup in our SIFT implementation.

4.2 Scalability Analysis

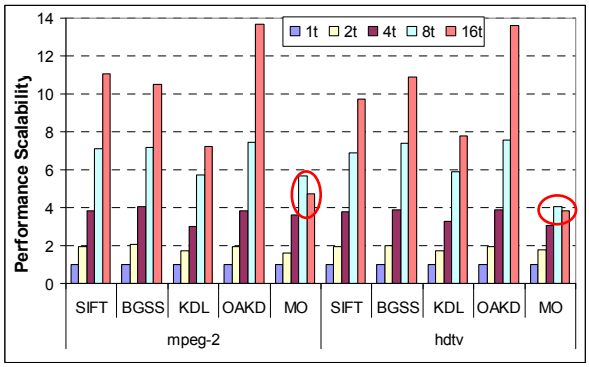


Figure 5 Performance Scalability

SIFT achieves a moderate speedup on the 16-core SMP system. Figure 5 shows the scaling performance of the entire application as well as each module on the 16-core SMP system. The whole application gets a speedup of 11x for MPEG-2 image and 9.7x for HDTV image respectively. To understand the limiting factors, we characterize the parallel performance from high level parallel overheads, e.g., synchronization penalties, load imbalance and serial regions. Figure 6 depicts the factors which limit the program scalability.

According to Figure 6, load imbalance and synchronizations occupy a large portion of the total execution time of parallel SIFT. The load imbalance problem mainly takes place in the KDL module, since the amount of key points detected varies significantly for different image scales, ranging from several hundreds to tens. As a result, the serious load imbalance reduces the scalability of KDL module. Since there are more key points detected in HDTV image, HDTV has less load imbalance problem than MPEG-2 as shown in

Figure 6, and the KDL module shows better scalability for HDTV (see Figure 5). With the increase of the thread number, load imbalance becomes more pronounced to limit the scaling performance.

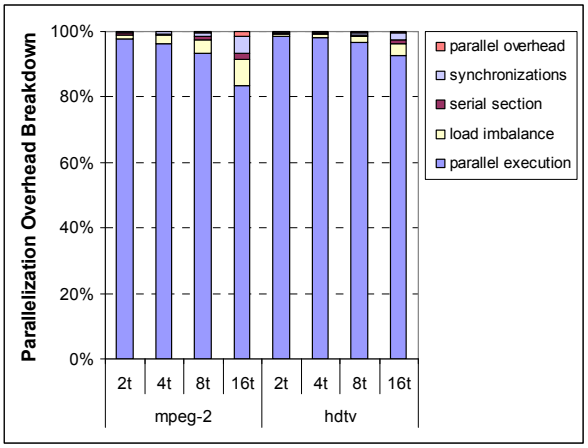


Figure 6 Execution Time Breakdown

Besides, synchronizations including barriers and locks affect the scalability too. They are mainly because of the synchronizations between the row and the column convolution in BGSS module and the critical section while merging different key point lists at the end of KDL module. In addition, serial section in Figure 6 is due to the data processing that cannot be parallelized between different iterations.

Figure 5 also shows that BGSS and OAKD modules scale well, but the scaling performance of KDL and MO modules are poor. As mentioned above, the KDL module suffers severely from load imbalance problem. The performance of the MO module actually degrades when the number threads goes from 8 to 16 (highlighted in Figure 5). It is because of its high memory bandwidth usage which will be discussed in the next subsection.

Based on all the general parallel overhead metrics shown in Figure 6, suppose the parallel region could scale perfectly SIFT should achieve the theoretical speedups of 13.6x for MPEG-2 and 14.4x for HDTV. They are much better than the real speedups we get from Figure 5. Therefore, we believe the scalability of SIFT is further limited by some other factors that will be discussed next.

4.3 Memory Behavior Analysis

Besides the general parallel performance metrics, the memory subsystem also plays an important role which affects the scaling performance severely. We now investigate why SIFT does not achieve the perfect scalability from the perspective of memory subsystem. We use Intel VTune analyzer to further analyze the memory behavior of SIFT on the 16-core SMP system. L1/L2 cache miss rates and system memory bandwidth requirement which are chosen as performance metrics are shown in Figure 8 and Figure 7.

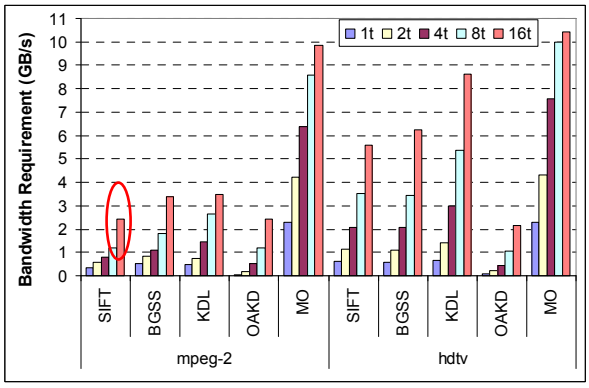
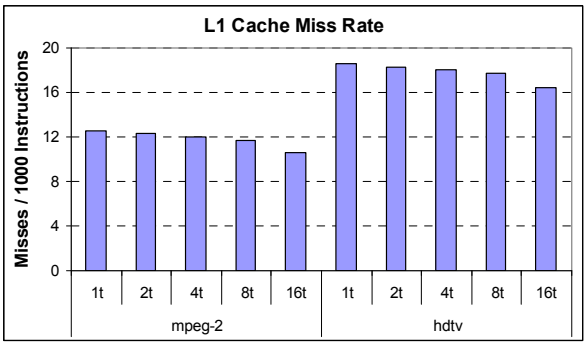


Figure 7 Memory Bandwidth Usages

4.3.1 Memory Bandwidth

Our first observation is that the average bus bandwidth does not limit the scalability of SIFT on the 16-core system. Figure 7 shows how the average FSB bandwidth utilization varies with different number of threads. For 16-thread, the bandwidth usage of SIFT for both images (2.4GB/s for MPEG-2 and 5.5GB/s for HDTV) is far below the peek bandwidth (33.3GB/s) that the system provides. Therefore, the scalability of SIFT is not limited by the average bus bandwidth of the 16-core SMP system.

Despite that SIFT is not bound to the average bandwidth usage, the scalability is limited by the instantaneous bandwidth usage caused by the burst memory transactions. The reason that the average bandwidth usage of SIFT is significantly lower than the peak bandwidth usage is because SIFT consists of different modules with different bandwidth requirements. Normally, the module with higher bandwidth requirement often limits the scalability of the entire application. As far as SIFT is concerned, apparently in Figure 7 the MO module has the highest burst bandwidth usage that it demands a bandwidth twice or even four times as much as the average bandwidth of the whole program. Although the peek bandwidth of the 16-core system is nearly 33.3GB/s, the FSB efficiency is just around 30%~40%. Thus the whole system can only provide a bandwidth of 10GB/s~13GB/s. Once the burst bandwidth usage of the application exceeds the system's capability, the performance will drop dramatically. That is why the speedup of MO of 16-thread is even worse than that of 8-thread. Definitely, MO module becomes the bottleneck and limits the whole application's scalability.



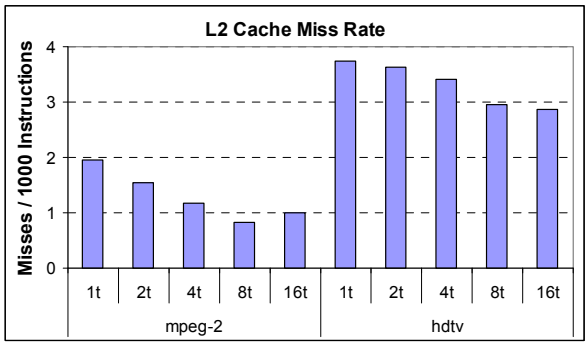


Figure 8 L1 and L2 Cache Miss Rates

4.3.2 Cache Performance

Additionally, there is a significant bandwidth usage increase — more than twice from 8 threads to 16 threads of SIFT for MPEG-2 (highlighted in Figure 7). We can find the reason in Figure 8. First, it is clear that mostly both L1 and L2 cache miss rates keep falling when the thread count scales. That is because a larger aggregated cache capacity is provided when the core number increases.

However, it is also observed that the L2 cache miss rate increases for MPEG-2 adversely from 8-thread to 16-thread. It is mainly because of the significant coherence misses. According to the SIFT algorithm there is a hierarchical parallel decomposition procedure, during which the downscaled image is read into all private L2 caches every iteration. Thereby significant coherence misses are incurred when SIFT scales, and it also aggravates the memory bandwidth requirement. The reason that the L2 cache miss rate keeps falling for HDTV from 8-thread to 16-thread is because HDTV has a much larger working set, thus less shared data will be contained by the L2 cache. As a result, coherence misses do not have serious impact on HDTV. We will discuss it later in Section 5. Nevertheless, we believe a shared LLC can help solve this problem.

5. PERFORMANCE ANALYSIS ON THE 64-CORE CMP

According to the performance analysis in previous section, for HDTV SIFT neither achieves the real-time performance, nor shows good scalability on the 16-core SMP system. With the prevalence of CMP and the steadily increasing number of cores in the near future, it becomes more important to harness the computing power on the large scale CMP. Thus, we model a 64-core CMP [2] with a shared LLC and high memory bandwidth to study its performance.

In the simulator, each core is in-order and has a private L1 data cache which is equipped with a hardware stride prefetcher, and all cores share the L2 cache. The cores are connected through a bi-directional ring, and the L2 cache is broken into multiple banks as well as distributed around the ring. Since each L2 bank has 16 MSHRs and can submit a request to memory every cycle, an available memory bandwidth of 55B/cycle or 110GB/s can be provided[2] with the 2GHz core frequency. Thus we suppose a high main memory bandwidth which does not artificially limit the scalability of the modules. Other parameters are listed in Table 1.

Table 1 Simulated System Parameters

Processor Parameters	
# Cores	1-64
Frequency	2GHz
Core width	2
Branch predictor	gshare, 2k entries
Memory Hierarchy Parameters	
Private (L1) cache	32KB, 8-way, 64B line
Shared L2 cache	16MB, 16 banks, 16-way (inclusion)
Coherence Protocol	MSI
Interconnect network	Bi-directional ring (40s tops)
Contentionless Memory Latencies	
L1 hit	3 cycles
L2 hit	18-58 cycles
Main memory access	298-338 cycles

5.1 Scaling Performance

As shown in Figure 9, SIFT shows much better scaling performance on the 64-core CMP than that on

[2] The 16MB LLC and 110GB/s memory bandwidth can be reached within 2~4 years, and the memory bandwidth is estimated as follows:

$$\text{Memory bandwidth} = \frac{\#\text{L2 banks} * \#\text{L2 MSHR / banks} * \text{Linesize}}{\text{memory latency}} \text{ [2]}$$

the 16-core SMP system for both MPEG-2 and HDTV images. In addition, HDTV even demonstrates better scalability (52x) than MPEG-2 (39x) in the case of 64-thread. Now SIFT can achieve 33FPS to meet the real-time constraint for HDTV image with 64 threads.

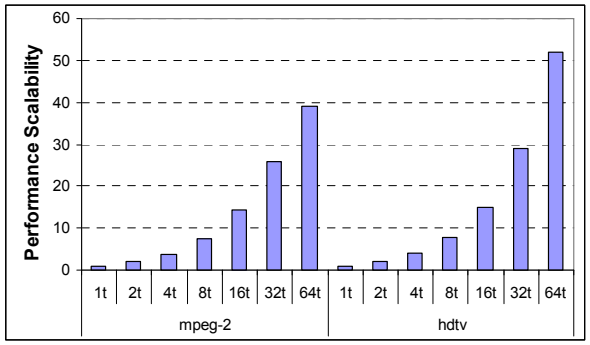


Figure 9 Scaling Performance

We further look at the factors that limit the scalability and why SIFT does not deliver linear scaling performance:

- ***Serial Sections*** The size of the serial sections in SIFT is reasonably small. It only occupies less than 2% of the total execution time for both MPEG-2 and HDTV images.
- ***Locking*** The lock overhead does not limit the scalability significantly either, as we have manually eliminated the lock overhead in the optimized parallel SIFT algorithm. It only occupies as much as 0.5% of the runtime.
- ***Parallel Overhead*** SIFT suffers from relatively high task queuing overhead. It is because as the number of tasks increases, the parallel region becomes smaller. Then the overhead of software dynamic task scheduling becomes large enough to affect the parallel speedup. Though software task queue overhead costs about 7% ~ 15% of the runtime for 64-thread, alternative hardware task distribution mechanism can help reduce the overhead effectively [18].
- ***Load Imbalance*** Load imbalance problem becomes even more pronounced to limit the scaling performance of SIFT on the 64-core CMP simulator and it is indeed the primary factor that limits the scalability. In the case of 64-thread, load imbalance takes about 15.9% of the total runtime for MPEG-2 and 8% for HDTV respectively. Apparently, since HDTV provides more tasks, it is better load balanced than MPEG-2 for 64-thread. That is why for HDTV SIFT gets better performance speedup.

Though SIFT dose not get linear speedup on the 64-core CMP simulator, it is obvious SIFT has better scalability than that on the 16-core SMP system. We can observe 14x speedup of 16-thread for both MPEG-2 and HDTV inputs. They are almost equal to the theoretical speedups considering the general scaling limited factors mentioned above. Therefore, we believe parallel SIFT is more cache/memory friendly on the 64-core CMP simulator.

5.2 Memory Behavior

In this section, we examine the memory behavior of the parallel SIFT algorithm on the 64-core CMP simulator in terms of working set sizes, data sharing behavior, on-die/off-die traffic and the effectiveness of prefetching.

5.2.1 Working Set Study

The cache miss rate curves versus cache capacity can provide us insights into how much temporal locality the program has and how effectively cache can reduce the bandwidth usage to next level memory hierarchy. The cache sizes corresponding to the knee points in the curves are commonly regarded as working set sizes of different level.

Figure 10 shows the shared L2 cache miss rate[3] for both MPEG-2 and HDTV images. We disable the hardware prefetcher since it may mask the working set size. We have the following observations:

First, SIFT has two level working sets. For single thread case, the first-level working set which is less than 32KB primarily consists of stack data only. While the second-level working set is around 12MB for MPEG-2 and 32MB for HDTV.

Second, the first-level working set size of SIFT scales with increasing thread count. For instance, the first-level working set of 64-thread which is around 2MB for MPEG-2 and 6MB for HDTV is much larger than that of 1-thread. This is because the stack data is private for each thread. Thus the first-level working set grows proportionally to the number of the threads.

However, the second-level working set size does not scale when we increase the thread count. In fact, it depends on the data parallel method we use, i.e., during different parallel sections, we always partition the image into several chunks and assign them to different working threads. Therefore, total amount of work does not change greatly when thread number scales. Thus, the second-level working set size remains. It is desirable to have a shared LLC for SIFT to accommodate its large second-level working set, since the shared LLC is able to avoid coherence misses and reduce the bandwidth requirement.

[3] This metric is Misses per Kilo Instructions (MPKI)

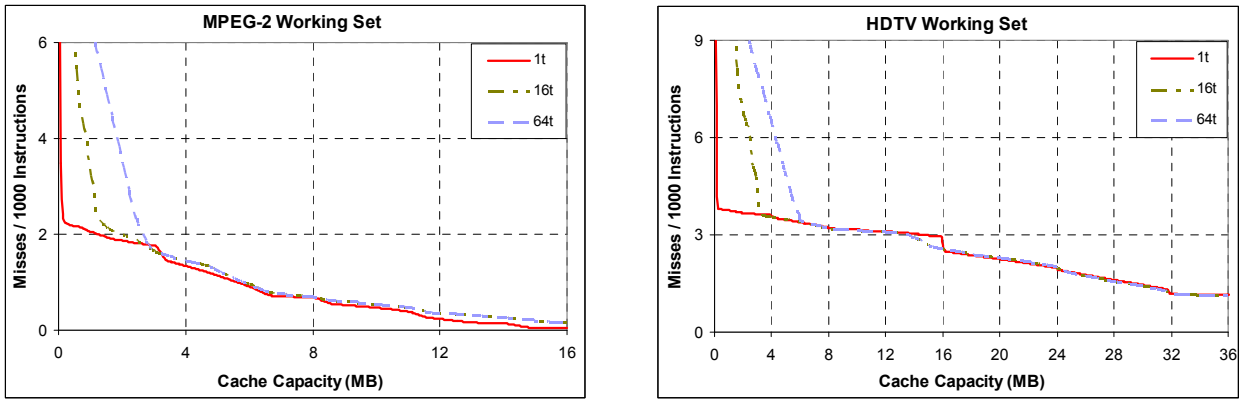


Figure 10 Working Sets of SIFT

5.2.2 Thread Sharing Behavior

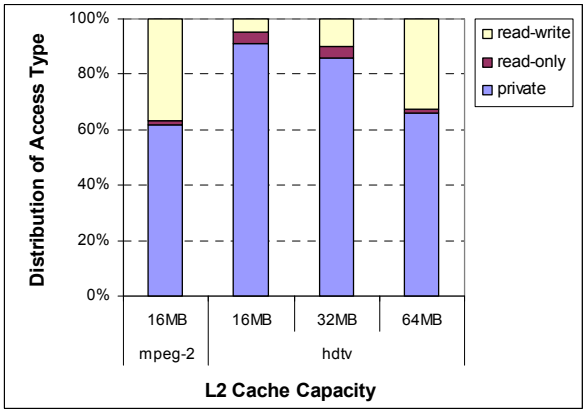


Figure 11 Distribution of Access Type

We study the data sharing behavior of SIFT to quantitatively identify the data access patterns on the 64-core CMP. Figure 11 shows the distributions of access type for 64-thread. We classify a shared cache line as either read-only shared or read-write shared. Read-only shared cache lines are those ones only read from but not written to, while read-write shared cache lines are those ones on which communications happen among threads.

Figure 11 reports that more than 35% of the data sharing for MPEG-2 is read-write shared. It confirms that parallel SIFT has high inter-thread communication as well as coherence misses that account for the increase of L2 cache miss rate from 8-thread to 16-thread for MPEG-2 on the 16-core SMP system. On the other hand, HDTV image only sees a small portion of read-write shared data as its second level working set is too large to fit into the 16MB L2 cache. Thus most private data that are used more frequently are contained. However, after enlarging the L2 cache capacity to 64MB, more cache lines become read-write shared as shown in Figure 11 since its second-level working set can fit into the L2 cache now.

5.2.3 On-Die Traffic and Off-Die Traffic[4]

Since working set and sharing behavior affect memory bandwidth seriously, we examine the on-chip and off-chip data communication next. It will provide us further insights into the memory behavior and bandwidth requirement of SIFT. Both on-die and off-die bandwidth are critical resources in scalable CMPs, since they are shared among a potentially large number of cores. An application will continue to scale only if sufficient bandwidth is available. The bandwidth results are shown in Figure 12.

On-die Bandwidth SIFT does not show a very high on-die bandwidth that even HDTV only requires 54GB/s bandwidth for 64-thread which is trivial compared to the 1TB/s ring based on-chip bandwidth.

Also, the on-die traffic rises steadily with the increase of thread count. This is because SIFT presents true inter-thread communication as we have mentioned earlier. In this way, the coherence traffic will increase the total on-chip bandwidth. However, it does not impact the performance speedup.

The on-die traffic can be classified into three components: useful prefetches, useless prefetches and demand accesses. Prefetching into the L1s can affect the on-die bandwidth usage. Overly prefetched cache lines will waste a great amount of bandwidth since they are not used at all or need to be re-fetched after evicted from L1 cache. Further, it may evict other useful data. But as Figure 12(a) shows, both for MPEG-2 and HDTV inputs the hardware prefetcher only creates very little useless prefetch traffic.

[4] We use the metric of Bytes/ALU here, the memory bandwidth can be estimated approximately as following: Memory bandwidth = #thread * IPC * off-die traffic * core frequency.

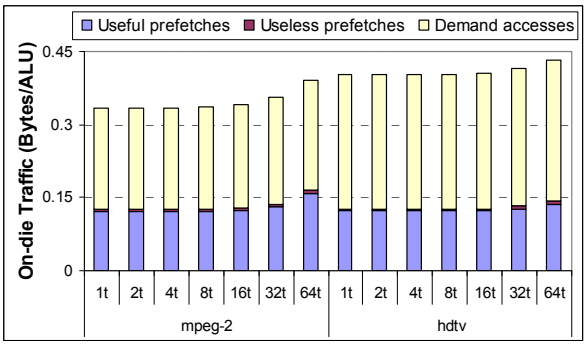
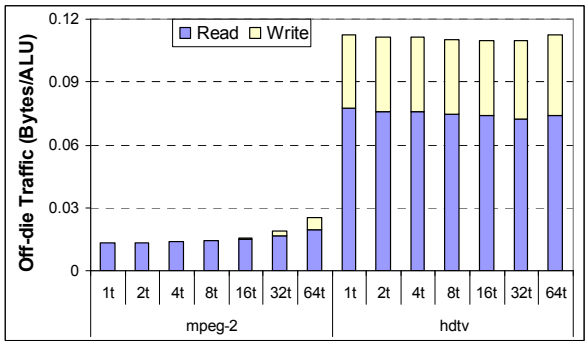


Figure 12(a) On-die Traffic



(b) Off-die Traffic

However, based on our performance data, the useful prefetch coverage that leads to L1 miss reduction is relatively low. It is because though SIFT is dominated by streaming accesses, its first level working set including the streaming data almost fits into the L1 cache. Thus the L1 miss rate is only around 1%, and most of the misses are caused by non-streaming accesses.

Off-die Bandwidth Figure 12(b) shows the data traffic between 16MB shared L2 cache and main memory classified by reads and writes. The off-die traffic for MPEG-2 follows the similar trend as the on-die traffic. While the off-die traffic for HDTV is insensitive to the number of threads. It is because of the coherence misses. Since for HDTV there are less sharing data contained in L2 cache as mentioned earlier, the coherence misses have relative little impact on the off-die traffic for HDTV while it affects the off-die traffic of MPEG-2 quite a lot. Furthermore, because most of the shared data are read-write shared, the write traffic for MPEG-2 increases accordingly while thread count scales.

SIFT has low off-chip bandwidth usage for MPEG-2 (less than 3GB/s) but relative higher bandwidth requirement for HDTV (14GB/s). It is because of their different working set sizes. Even though, it will not limit the scaling performance of SIFT since the memory bandwidth is far from saturating on the 64-core simulator. Moreover, as opposed to the 16-core SMP system the bandwidth requirement of 16-thread on the 64-core CMP simulator is much lower (less than 4GB/s for HDTV image). It demonstrates the benefits of the shared LLC.

6. Related Work

Different from our study that mainly focus on the general purpose multi-core systems, several SIFT algorithm accelerations were proposed on GPU [17] [19] in order to make SIFT run in real-time. Sinha et al. [19] implemented SIFT algorithm on GPU. The results show that their implementation can only extract about 800 key point features from a 640x480 video image at 10 FPS. Heymann et al. [17] proposed another SIFT implementation on GPU that can be applied to image sequences with 640x480 pixels at a speed of 20 FPS.

Though the performance of SIFT on GPU are attractive, they are still much slower than a general purpose 64-core CMP, which is capable of delivering real-time performance even for the HDTV images (1920x1080).

7. Conclusion

In this paper, we parallelized and optimized the SIFT algorithm. Also we conducted a detailed performance characterization both on a 16-core SMP system and on a 64-core CMP simulator. We found the optimization techniques such as SIMD and cache-conscious optimization are effective, which can bring an 85% performance gain for SIFT at most. Regarding the parallel performance, load imbalance affects the scaling performance of SIFT seriously on both systems. In addition, SIFT suffers from high burst memory bandwidth requirement that reduces its scalability on the 16-core SMP system. However, the memory bandwidth demand is not high for the 64-core CMP simulator due to its large shared LLC and high available memory bandwidth. SIFT presents intensive inter-thread communication due to the tremendous read-write sharing. A shared LLC can help eliminate the coherence traffic significantly.

The characterization of SIFT can help us understand the SIFT program better and gives us insights into designing future systems.

References

[1] A. Jaleel, M.Mattina, and B. Jacob, "Last-Level Cache (LLC) performance of data-mining workloads on a CMP-A case study of parallel bioinformatics workloads", *in 12th HPCA, February* 2006.

[2] C.J. Hughes, R. Grzeszczuk, E. Sifakis, D. Kim, S. Kumar, A.P. Selle, J. Chhugani, M. Holliman, and Y.K. Chen, "Physical Simulation for Animation and Visual Effects: Parallelization and Characterization for Chip Multiprocessors*", in 34th ISCA,* 2007.

[3] C. Liu, A. Sivasubramaniam, M. Kandemir, "Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs", *in 6th HPCA,* 2004.

[4] David G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", *International Journal of Computer Vision, Volume 60, Number 2*, 2004.

[5] F. Estrada, A. Jepson, and D. Fleet, "local Features Tutorial", *http://www.cs.toronto.edu/~jepson/csc2503*, 2004.

[6] J.D. Aalehi, J.F. Kurose, and D.F. Towsley, "The Effectiveness of Affinity–Based Scheduling in Multiprocessor Networking", *in INFOCOMM'96,* 1996.

[7] Jung Ho Ahn Erez, M. Dally, W.J., "Scatter-add in data parallel architectures", in *11th HPCA,* 2005.

[8] L. Hur, C. Lin, "Memory Prefetching Using Adaptive Stream Detection", *in 39th MICRO*, 2006.

[9] IBM Corp. PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual.

[10] Intel Corporation, *Intel 64 and IA-32 Architectures Optimization Reference Manual*, 2006.

[11] Intel Corporation, "Intel Threading Analysis Tools", in *http://www.intel.com/software/products/Threading*.

[12] Intel Corporation, "Intel VTune Performance Analyzer", *http://www.intel.com/software/products/vtune*.

[13] Iryna Skrypnyk, and David G. Lowe, "Scene Modeling, Recognition and Tracking with Invariant Image Features", in *ISMAR'04*, November 2004.

[14] *OpenMP Application Program Interface*, Version 2.5, May 2005.

[15] Qi Zhang, Yurong Chen, Yimin Zhang and Yinlong Xu, "SIFT Implementation and Optimization for Multi-Core Systems", 10th Workshop on APDCM, February 2008.

[16] Shahbahrami, Asadollah; Juurlink, Ben; Vassiliadis, Stamatis, "SIMD Vectorization of Histogram Functions", *ASAP,* 2007.

[17] S. Heymann, K. Müller, A. Smolic, B. Froehlich, and T. Wiegand, "SIFT Implementation and Optimization for General-Purpose GPU", in *WSCG'07*, January 2007.

[18] S. Kumar, C.J. Hughes A. Nguyen. "Carbon: Architectural Support for Fine-Grained Parallelism on Chip Multiprocessors", *in ISCA*, 2007.

[19] Sudipta N. Sinha, Jan-Michael Frahm, Marc Pollefeys, and Yakup Genc, "Feature Tracking and Matching in Video Using Programmable Graphics Hardware", *Machine Vision and Applications*, March 2007.