Classifier Grouping to Enhance Data Locality for A Multi-Threaded Object Detection Algorithm

Bo-Cheng Charles Lai, Chih-Hsuan Chiang, Guan-Ru Li
Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan

Abstract — **Object detection has become an enabling function for modern smart embedded devices to perform intelligent applications and interact with the environment appropriately and promptly. However, the limited computation resource of embedded devices has become a barrier to execute the computation intensive object detection algorithm. Leveraging the multi-threading scheme on embedded multi-core systems provides an opportunity to boost the performance. However, the memory bottleneck limits the performance scalability. Improving data locality of applications and maximizing the data reuse for on-chip caches have therefore become critical design concerns. This paper comprehensively analyzes the memory behavior and data locality of a multi-threaded object detection algorithm. A novel *Classifier-Grouping* scheme is proposed to significantly enhance the data reuse for on-chip caches of embedded multi-core systems. By executing a multi-threaded object detection algorithm on a cycle-accurate multi-core simulator, the proposed approach can achieve up to 62% better performance when compared with the original parallel program.**

Keywords - data locality; object detection; parallel processing; multi-core; embedded device;

I. INTRODUCTION

Intelligence has become an essential feature for modern smart embedded devices. These devices are able to recognize the surrounding environment through sensing various types of stimuli, including vibration, orientation, temperature, sound, images, video, and etc [1]. With the awareness of the surroundings, a smart device can make decision and react intelligently to specific stimuli or events in real-time or within acceptable latencies. For example, modern tablets can sense the orientation of the device through gyroscopes and adjust the screen orientation for users [2]. The future smart application can perform even higher levels of intelligence, for example, to automatically identify different users, or even search the customer background in real-time during a business conference [3].

Among different types of stimuli, images and real-time video have the richest information about the environment. Images and video contain information close to the level of human eyes. Many embedded devices have applied image sensing techniques and perform functional enhancement for applications. For example, most of the digital cameras can nowadays identify human faces in the target zoom and tune the best focus for the picture. A game console can recognize the movement of players through the embedded image sensors, and let the player control the console without a physical controller [4]. A high-end car is even equipped with cameras to automatically monitor the current driving direction. When the direction is deviating from the main path, a waning message of the potential danger will be sent to the driver before an accident happens [27]. All of these intelligent applications require a capability to recognize the existence as well as the location of the target object in the current monitor screen area. Some real-time applications, such as the lane departure warning system of a car, even require a prompt recognition thus the system can feedback the appropriate action in time.

However, processing the image data and finding the target object require intensive data computation. It is estimated to take about 2 seconds to recognize an object in an image of 720*576 pixels on a 2.33GHz Intel® Core™ 2 Quad processor [17]. Even with such a powerful general purpose processor, the execution time is 50 times slower than the requirement of a real-time application (processing at least 25 frames per second). This computation requirement poses a more stringent barrier for a portable embedded device, which is highly constrained in computation resources.

Parallel processing and multi-core architectures are considered as a solution to achieve high performance and efficient computation. They have become the mainstream to the design of modern computing systems. Embedded processor vendors, such as Tilera [5], ARM [6], MIPS [7], are proposing multi-core architectures. Even the desktop processor vendors, such as Intel and AMD are planning multi-core products for embedded and mobile applications. The new parallel embedded processors present opportunities to boost the raw computing capability and achieve energy efficient execution. However, the limited off-chip memory bandwidth and long access latency have imposed a limitation to the system performance [8]. Efficient usage of the on-chip memory, especially the cache of processors, has therefore become a critical design issue to achieve performance scalability of embedded multi-core systems.

The main contributions of this paper can be categorized into three folds. First, this paper comprehensively investigates the memory access behavior of a multi-threaded Viola-Jones-based [24] object detection algorithm on an embedded multi-core system. Second, from the analysis results, we have concluded that optimizing the locality of classifier features is more effective than the locality of the integral image data. Third, based on this observation, a new design scheme, Classifier-Grouping (referred as CG in this paper), is proposed to enhance the data locality on the local processor cache.

Without affecting the accuracy and quality of the original object detection algorithm, *Classifier-Grouping* clusters the execution of appropriate numbers of classifiers and enhances the data reuse of the local cache on an embedded multi-core system. The improved data locality maximizes the data reuse for on-chip caches and effectively avoids the off-chip memory bottleneck. The overall system performance can then be improved significantly. By running the optimized multi-threaded object detection algorithm on an ARM-based cycle-accurate SMP (Symmetric Multi-Processing) simulator [10], we are able to improve the system performance up to 62% when compared with the reference parallel implementation.

This paper is organized as follows. Section II discusses the related work. A multi-threaded Viola-Jones-based object detection algorithm is introduced in Section III. Section IV shows the cycle-accurate SMP simulation platform used in this paper. Section V comprehensively analyzes the memory behavior of the multi-threaded object detection algorithm. Based on the analysis results, Section VI proposes Classifier-Grouping, a new design scheme of the algorithm which optimizes the data locality on embedded multi-core systems. The experimental results are also illustrated in this section. Section VII draws the conclusion and future work.

II. Related Work

Object detection is an indispensible function for smart embedded devices. By extracting the features in a sensed image, it is among the first step for a device to understand the surrounding environment. The Viola-Jones algorithm is one of the widely used object detection schemes [9]. It was first proposed to detect human faces in an image. Due to the high accuracy and fast computation, the algorithm has also been extended to detect other objects in an image or video, such as hands, eyes [13], pedestrians [14], and cars [15]. To achieve fast object detection, many research efforts have focused on enhancing the performance by using specific hardware modules. The specifically designed hardware exploits the algorithmic parallelism and speeds the critical arithmetic operations by hardware accelerators [16][20][21][22].

The capital expenditure of NRE (Non-Recurring-Expense) is increasing significantly with the advances of the semiconductor technology. Without a mass volume of production, a pure hardware solution is becoming less attractive economically. Moreover, the less flexible hardware design is difficult to adapt to the changes and specific operation requirements of the ever changing features of the future applications, especially for the intelligent applications. However, the applications of the future systems do not accept a compromised performance provided by a traditional single-core programmable platform. The recently emerging embedded multi-processors leverage on the scalability of the Moore's Law and can boost the performance by exploiting the parallelism of applications. The enhanced performance with the programmable feature make the multi-core system a cost attractive solution in designing future embedded intelligent applications.

Chen‘s research [17] is among the first to explore the algorithmic parallelism of the Viola-Jones-based detection algorithm on programmable processors. The potential parallelism of the Ada-boost algorithm was analyzed and executed on multi-core systems with 4 to 8 processors. A 5.5X performance enhancement was demonstrated by adopting a hybrid scheme of both coarse-grained and fine-grained TLP (Thread Level Parallelism). Chen et al. [25] implemented the OpenCV object detection algorithm on a Cell processor and obtained 19% of performance improvement. Chiang [18] investigated the characteristics of different parallelism levels of a Viola-Jones algorithm and proposed a three-staged parallelization scheme to improve the load balance of the algorithm. A 37X performance improvement is achieved based on the proposed scheme. However, Chiang's work mainly focused on the algorithmic parallelism on a multi-core system and did not take into account the performance degradation caused by the memory bottleneck.

Data locality optimization is a critical design issue for computing systems and has been studied for decades [11]. However, most of the previous research works focused on the locality optimization for the single core system. Locality issues of multi-core systems are recently emerging as essential design concerns when the parallel platforms become the mainstream of the computing architecture [12]. In a shared memory multi-core system, the design needs to be balanced between parallelism and locality in order to achieve the best overall performance.

This paper differs from the previous work in two aspects: (1) the target platform of this work focuses on embedded multi-core systems. Each core is a simple single issue RISC programmable processor with relatively small on-chip caches; (2) this work concentrates on the data locality optimization for the parallel object detection algorithm and proposes a design to improve the memory access behavior as well as overall performance.

III. Parallelize An Object Detection Algorithm

This paper focuses on the Viola-Jones object detection algorithm [24]. The Viola-Jones algorithm was first designed for face detection on a still image. It features low computation complexity and high recognition accuracy, and was extended to support various detection applications of different target objects.

This section delves into the detailed flow of this algorithm and the multi-threaded implementation on a programmable computer. The first part of this section introduces the main algorithm flow of the Viola-Jones algorithm and discusses the functions and properties of each operation. The second part shows the sequential implementation of the reference design adopted form OpenCV [19]. The third part discusses the inherent parallelism of the algorithm and a load-balanced multi-threaded implementation.

A. Viola-Jones Algorithm

As shown in Fig.1(a), the main purpose of an object detection algorithm is to decide the existence and position of the target object by checking specific visual features in the current image. Fig.1(a) illustrates the flow of the Viola-Jones algorithm. After the target image is loaded into the detection system, the image is checked by different sizes of scan windows in order to identify the target objects of various sizes. Each scan window will inspect every position in the target image by applying the cascaded classifiers (Fig.1(b)).

Viola-Jones algorithm is the first to utilize *AdaBoost* as part of the learning algorithm for object detection. AdaBoost uses a cascade of classifiers to implement an efficient and accurate detection mechanism [9]. The image in each scan window passes through a series of classifiers during detection. A number of visual features are selected to represent the target object. Each classifier performs the detection of a part of the features. As shown in Fig.1(b), a small number of weak classifiers with similar features form a strong classifier, and several strong classifiers are cascaded into a complete object detector. A strong classifier at a later stage contains more weak classifiers to provide more rigorous feature checks. This cascaded structure rapidly and efficiently rejects most negative window positions while keeping almost all the positive ones.

The features of an object within each scan window is evaluated based on the *Haar-like feature* [9], where the value of each feature is obtained by the sum of the pixel values in the white rectangles of the feature minus the sum of pixel values in the black rectangles (Fig.1(c)). The outcome of each Haar-like feature in a classifier stage is computed and accumulated. When all the features in a stage are computed, a *stage threshold value* is used to determine whether the sample in the current scan window is a successful candidate to move on to the next stage or not.

The Viola-Jones algorithm applies the *integral image* method to rapidly compute the Haar-like features. The integral image method was originally introduced to perform the digital image processing [23]. By using the integral image method, the computation of the (weighted) intensity difference between two to four rectangles can be efficiently obtained [9]. This scheme provides a fast method to compute the target features in a rectangular sub-region area.

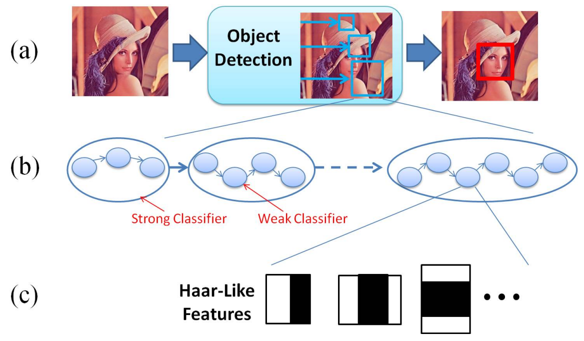

Figure 1. (a) Flow of the Viola-Jones object detection algorithm; (b) cascaded structure of classifiers; (c) Haar-like features within each classifier.

B. *Sequential Implementation of Viola-Jones Algorithm*

Fig.2 illustrates the pseudo code of the sequential reference design of the Viola-Jones algorithm. The reference design in this paper is adopted and modified from OpenCV library [19].

```
1:  for all WZ                          // WZ: window size
2:    Resize image;
3:    Integral image;
4:    for all WP                        // WP: window position
5:      Detect {
6:        for all SC                    // SC: strong classifier
7:          for all WC                  // WC: weak classifier
8:            If failed, label the position as negative; jump to the next WP;
9:        If passed all the SCs, label the position as positive;
10:     } // end of Detect
```

Figure 2. Pseudo code of the sequential Viola-Jones algorithm.

As shown in Fig.2, the sequential implementation of Viola-Jones algorithm can be divided into three parts. **(1) Resize.** The implementation uses the fixed-size scan window with a well-trained classifier library for the AdaBoost algorithm. Since the scan window size is fixed, an input image needs to be resized into different resolutions. **(2) Integral.** This part calculates the integral information for each image position. This integral information enables fast evaluation of the Haar-like features in the Detection part. **(3) Detection.** By moving the scan window through the image, the image area covered by the scan window is sent into the cascaded classifier structure to decide the existence and location of the target object.

The **Detection** block contains a list of cascaded strong classifiers. Each strong classifier is composed of a series of weak classifiers. When the scan window steps to a new position, the image data covered by the scan window will be checked by these cascaded classifiers. If the image data passes all the classifiers, this window position will be marked as positive, which means there exists a target object at this position. Otherwise, the cascaded classifiers will reject a position as soon as it fails the check.

Fig.3 illustrates a more detailed pseudo code for the Detection block. During the execution of weak classifiers, the feature parameters of each weak classifier are loaded. These parameters include information of the Haar-like features, weighting factors for each Haar-like rectangle, weighting factor of the weak classifier, and the threshold of the weak classifier. Based on the OpenCV library, each weak classifier contains two to three Haar-like rectangles.

```
1:  Detect {
2:    for all SC {                      // SC: strong classifier
3:      for all WC {                    // WC: weak classifier
4:        load parameters of WC;        // position, Haar-like features,
5:                                      // weighting factor, threshold,
6:        load Integral data of the feature positions;
7:        calculate the weighted sum of the feature;
8:        stage_sum += WC.alpha [weighted sum >= WC.threshold];
9:      } // end of WC
10:
11:     if ( stage_sum > SC.threashold )  return positive feedback;
12:     else  return negative feedback;
13:   }  // end of SC
14: } // end of Detect
```

Figure 3. Pseudo code of the detailed flow of Detection block

Since OpenCV is a reference design running on a general purpose processor, it implements a lot of floating point operations to perform the object detection. However, floating point computation is an expensive operation on an embedded processor. This is not only because the operation takes more cycles to finish, but also consumes more energy and chip area. Thus, if possible, most embedded applications would remove the floating point computation and substitute it with fixed-point operations. The downside of using the fixed-point scheme is the degradation of computation precision. However, Viola-Jones algorithm is robust to the lower computation precision because of its multi-resolution characteristics. The same image will be resized and checked under different resolutions, thus the degradation of the computation precision does not cause negative decisive impact on the final results. The reference design used in this paper is a fixed-point implementation of OpenCV design. After a careful tuning, the fixed-point design shows the same recognition accuracy as the original floating point implementation.

C. A Multi-Threaded Design of Viola-Jones Algorithm

To take advantage of the computation power enabled by an embedded multi-core system, the Viola-Jones algorithm needs to be parallelized into multiple threads. Since this paper focuses on a shared-memory symmetric multi-processor platform, the key design concerns of parallelizing an application can be categorized into two folds. The first concern is the parallelization granularity, which decides how fine-grained a designer would like to expose the inherent parallelism. The second concern is how much overhead, for both computation and synchronization, will be generated when parallelizing an application. This section will give a detailed discussion on these two design concerns.

The parallelism of the Viola-Jones algorithm can be exposed at different algorithmic levels. For example, designers can process different window sizes by different threads concurrently (Window Size Level). The algorithm can also be parallelized by simultaneously executing each weak classifier with a different thread (Weak Classifier Level). The lower parallelism level, such as Weak Classifier Level, gives finer granularity of parallelism. This scheme creates more threads and gives a better balance among the execution loads of different concurrent threads. Well balanced task loads can potentially utilize the parallel computation platform more efficiently. Processors do not need to stay in the idle mode and wait for the finish of a critical thread with long execution time. However, having more threads in a system increases overhead of thread creation. It also causes a higher amount of synchronization between threads and demands higher memory and interconnection bandwidth. On the other hand, the higher parallelism level, such as Window Size Level, generates fewer concurrent threads and imposes lower synchronization overhead. However, the performance of a design with high parallelism level could suffer from the imbalanced task loads.

To achieve superior performance, a multi-threaded design needs to strike the balance between the parallelism granularity and the synchronization overhead. This paper adopts the *3-Stages Hybrid Scheme* parallel implementation proposed in [18]. The parallelism is exposed from two methods, including the functional stages of the algorithm flow and the data processing of each stage. The functional stage of the Viola-Jones algorithm can be intuitively identified as the three main functional blocks shown in Fig.2, including Resize image, Integral image, and Detection block. These three stages have clear boundaries which separate the computation and the associated data. Therefore it is a reasonable design choice to process these stages concurrently in a functional pipelining scheme.

The second method to expose the parallelism is the data processing within each functional block. An obvious parallelism exists in the data domain of the functional block, where each block needs to process different sizes of the images. There is no data dependency between computations of different image sizes, and thus these computations can be executed concurrently by multiple threads.

Fig.4 illustrates the 3-Stages Hybrid scheme. The first stage contains a multi-threaded version of the **Resize** block. An image is split into several sub-image chunks. The resize task of each sub-image chunk is performed by a thread. In this way, multiple threads can perform the resize of an image concurrently. The same scheme is applied to different sizes of images. To achieve better task load balance, a larger image is divided into more sub-images and executed by more concurrent threads. The second stage processes the **Integral** blocks of different image sizes concurrently with multiple threads. The third stage performs the parallel execution of the **Detection** block. Similar to the technique used in the parallel Resize block, a larger image is divided into more sub-images and executed by more concurrent threads. The 3-Stage Hybrid scheme finds a proper design point between the trade-off of load balance and synchronization overhead, and achieves a better overall performance.

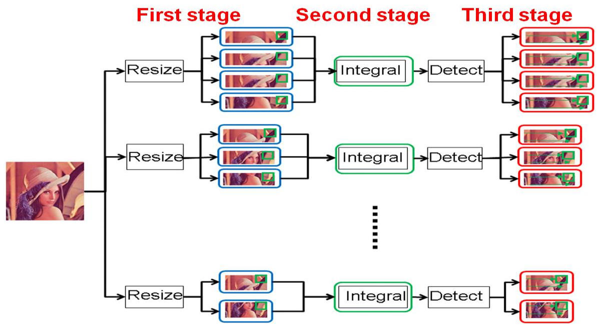

Figure 4. The 3-Stages Hybrid scheme of the parallel Viola-Jones algorithm.

IV. Experimental Setup: A Cycle Accurate SMP Simulator

The experiments of this paper are executed on a cycle-accurate multi-threaded shared-memory SMP simulator [10]. The simulator performs HW/SW co-simulation of both the multi-threading SW library and the cycle-accurate SMP hardware model. This platform returns a precise evaluation of the overall system characteristics and performance.

Fig.5 illustrates the organization of the simulator. The multi-threading library is based on QuickThread [26]. This light-weighted library including the boot code fits in under 2KBytes of object codes. There are four generic APIs (Application Programming Interface), including *stp_create()*, *stp_start()*, *stp_yield()*, and *stp_stop()*, which enable the creation of threads, execution of threads, yielding of a thread, and termination of a thread respectively. These APIs facilitate the control of multiple threads executing on a parallel platform.

After creation, all the threads are managed by a FIFO queue. The queue is implemented in the shared memory space, and can be accessed by all the processors. The FIFO queue requires an atomic access and is protected by a spin-lock synchronization mechanism. Each processor can create and add new threads to the tail of the queue. When a processor finishes executing the current task, it will request the next task from the head the queue.

The processing core models a single-issue ARMv5 architecture with no floating point unit. Each processor has its own data cache and instruction cache. The configurable cache organization enables easy exploration of different cache line sizes, block sizes, and associativity. The interconnection implements a single-transaction shared-bus. Although not scalable, the shared-bus scheme is still widely used in today's embedded multi-core systems. The latencies of bus transactions and memory accesses are also configurable. The experiment can adjust the shared-bus latencies to reflect the salability issue when the number of processors is increasing.

All the processors are sharing the same memory space. The cache coherence is implemented as a simple snooping-based protocol. The important system parameters used in the simulator are shown on the right hand side of Fig.5. In this paper, each processor is assumed to run at 250MHz. The nominal size for each local cache is 8KBytes. Each cache block has 16 cache lines, and supports full associativity.

The maximum number of processors used in this paper is sixteen. The arbitration of the single-transaction bus takes 1 processor cycles (4ns), which is feasible with today's advanced semiconductor technology [28]. The signal traversing on the shared-interconnection takes 2 processor cycles (8ns). This parameter could also be used to represent the latencies required for the data to traverse the path of on chip network, if the processors are connected by a NoC (Network on Chip).

The access latency of the main memory takes 7 cycles (28ns), which is approximately the latency of the modern SDRAMs [29]. The ARM-based cycle accurate simulator can well represent the system characteristics of the majority of embedded multi-core systems.

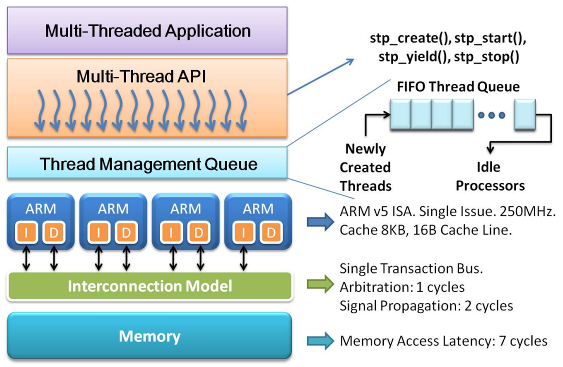
Multi-Threaded Application
Multi-Thread API
Thread Management Queue
ARM
I
D
Interconnection Model
Memory
stp_create(), stp_start(), stp_yield(), stp_stop()
FIFO Thread Queue
Newly Created Threads
Idle Processors
ARM v5 ISA. Single Issue. 250MHz. Cache 8KB, 16B Cache Line.
Single Transaction Bus. Arbitration: 1 cycles Signal Propagation: 2 cycles
Memory Access Latency: 7 cycles

Figure 5. The SW/HW organization of a multi-threaded shared-memory SMP simulator. The system parameters are shown on the right hand side.

V. Data Locality of the Detection Block

This paper focuses on the data locality optimization for the **Detection** block. Based on the analysis results, a new design scheme, Classifier-Grouping (CG), is proposed to enhance the data reuse of on-chip caches.

As illustrated in Fig.2, the implementation of Viola-Jones algorithm can be divided into three blocks, including **Resize**, **Integral**, and **Detection**. According to [18], **Detection** occupies around 70% of the overall runtime. Resize and Integral parts only take around 30% of the overall execution time. Note that this ratio shows up differently when applying on different parallel platforms. For example, [25] has shown that the Detection block has taken up 95% to 98% of the overall execution time. The main reason behind the disparity is the difference of the implementation schemes and the underlined parallel platforms. However, all the related studies have concluded that the Detection block is the most significant time consuming part in the overall application, and should be the main focus of the further optimization.

Another reason of concentrating on the Detection block is its inherent high data locality, which is not the case for the other two blocks (Resize and Integral). From the parallel implementation scheme described in section III, the Resize block reads the original image data and resizes the image into different resolutions. The original image data will be used only one time by each thread. The Integral block takes a resized image and calculates the integral information for different positions. Again, the resized image data will be read only once by each thread. The main performance enhancement of these two blocks comes from the exploitation of the high data level parallelism.

The Detection block not only has the high data level parallelism, it also possesses significant amount of data reuse during the computation. When executing on an embedded processor, the Detection block has contained two types of data locality. The first type is the feature information used by each classifier, including types of Haar-like features, position in the scan window, weighting factors, threshold, and etc. The second type is the image data that would be processed by cascaded classifiers. The same data points of an integral image within a scan window could be reused by different classifiers during the detecting procedure. However, optimizing one type of data locality could conflict the data reuse of the other type.

For example, the reference implementation of Fig.2 can potentially take advantage of the data locality of the integral image data. The Detection block fixes on the same scan window and checks this window by the cascaded classifiers. During the detecting procedure, the loaded integral image data will be stored in the local cache of a processor. If this scan window passes the current classifier, the next classifier might need and therefore reuse the same data points in the cache. However, due to the relatively small cache sizes of an embedded processor (8KBytes in this paper), the feature information of the already used classifiers could be flushed when the scan window moves to a new position and starts loading the new integral image data.

Due to the conflict characteristic of these two types of data locality, we have to evaluate the impact of each type and choose a more effective one.

A. *Analysis of Two Types of Data Locality within Detection Block*

The observation shows that the cascaded classifier structure rapidly and efficiently rejects most negative window positions while keeping almost all the positive ones. From our experiment, around 75% of the scan window positions failed within the first three strong classifiers, which contain a total of 40 weak classifiers. Almost 90% of the scan window positions failed within the first five strong classifiers, which contain 112 weak classifiers. This observation gives a hint that it could be more effective to exploit the data locality of classifiers rather than the integral data points. The following paragraphs will detail the analysis of these two types.

Each weak classifier includes two to three rectangular Haar-like features. Viola-Jones algorithm uses the integral image to efficiently compute the target features. Thus each classifier only loads the corner integral data points of a Haar-like feature rectangle and computes the intensity difference between sub-rectangles. The possibility that the corners of these rectangles fall on the same data position of an integral image is fairly low. Fig.6 illustrates this characteristic.

Let us use a simple example to illustrate this case. In the reference implementation, each resized image will be scanned by a 20x20 pixel scan window (total 400 pixels). Assume a scan window has passed the first five strong classifiers, which has used a total of 952 integral data points. Each integral data point, in average, will only be used a little bit more than two times. This data access characteristic cannot take a full advantage of the local data cache and has concluded that it can only expose very little data locality when focusing on the reuse of the integral image data.

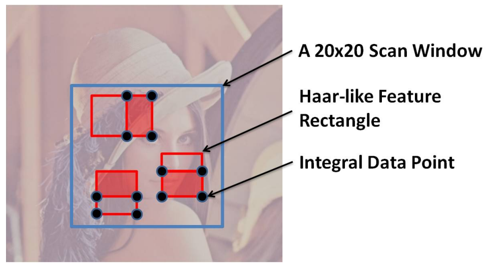

Figure 6. Haar-like feature rectangles in a scan window.

The feature data of classifiers, however, has a much better data locality than the integral data points. Recall the property that most of the scan window will be rejected within the first three strong classifiers. If we can somehow keep the feature data of the first three weak classifiers in the local cache of a processor, almost all the scan window can benefit from reusing the data of classifier features in local caches. For a 512x512 image, there are more than 290K scan window positions. The system performance can be enhanced significantly if most of these scan window positions can reuse the classifier feature data in local caches.

VI. CLASSIER-GROUPING: A SCHEME TO ENHANCE DATA LOCALITY

Based on the analysis from the previous section, we have proposed a new design scheme to enhance the data locality for an embedded multi-core system. The first design scheme is to change the order of the execution loop. It is referred as Loop-Changing (LC) scheme in this paper. The strategy of the LC scheme tries to avoid the classifier feature data from being flushed by the subsequently loaded integral image data. This strategy has enhanced the overall system performance and proved the effectiveness of the conclusion of our data locality analysis. Based on the LC scheme, a Classifier-Grouping (CG) scheme is then proposed to further change the program organization and bind the execution of the first three strong classifiers. CG improves the data reuse and provides a performance enhancement up to 62%.

A. *Loop-Changing Scheme to Enhance Data Reuse*

Fig.7 shows the pseudo code of the LC scheme. Different from the original algorithm flow in Fig.2, the LC design moves the loop of WP, which changes the position of the scan window, to the place after the loop of SC (Strong Classifier).

The LC design improves the locality of the feature information of weak classifiers. Moving the WP loop after the SC loop can reduce the possibility for the already-cached feature information being replaced by the subsequently loaded data right after the usage of the current strong classifier.

```
1: for all **WZ**          // **WZ: window size**
2:   **Resize** image;
3:   **Integral** image;
4:   **Detect** {
5:     for all **SC**       // **SC: strong classifier**
6:       for all **WP**   // **WP: window position** (New position of WP loop)
7:         for all **WC**  // **WC: weak classifier**
8:           If failed, label the position as negative; jump to the next WP;
9:           // processor can better reuse the WC data stored in the cache
10:          // which significantly increases the data locality
11:    If passed all the SCs, label the position as positive;
12:  } // end of **Detect**
```

Figure 7. Pseudo code of the Loop-Changing (LC) scheme.

Fig.8 shows the runtime comparison between the reference parallel scheme (3-Stage Hybrid scheme) and the LC design scheme. The cycle-accurate simulation has been performed for systems with different numbers of processors (x-axis of Fig.8). The LC design has better performance at all the multi-core schemes. This is because the numbers of external memory accesses are reduced significantly due to the better data locality at the local cache. The 16-processor scheme has the maximum performance enhancement of 58%. This is mainly because the potential performance enhancement of the 3-Stage Hybrid enabled by more processors is compromised by the enormous memory access time. The memory bottleneck becomes the limiting factor of the system performance. Hence the performance stops improving when there are more than eight processors. However, the LC design significantly reduces the number of memory accesses. The performance continues to scale when there are more processors (8, and 16 processors).

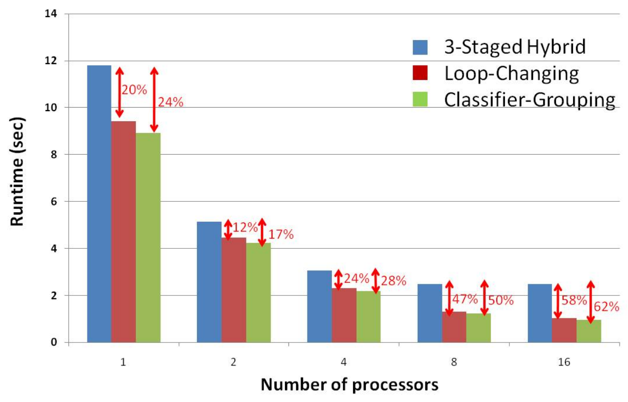

Figure 8. Performance comparison between the different design schemes (3-Staged Hybrid, Loop-Changing, Classfier-Grouping)

B. *Classifier-Grouping Scheme to Further Enhance Locality*

The results of LC design encourage us to further bind the execution of the first couple of weak classifiers to gain more benefit from the locality of classifier features. It is called Classifier-Grouping (CG) scheme. Fig.9 shows the pseudo code of an example of the CG scheme, which binds the execution of the first three strong classifiers. The basic idea is to execute the first three strong classifiers for all the window positions (line 5 to 8 in Fig.9). This way, the features of the first three strong classifiers can be reused extensively by the scan windows at all the window positions. After all the window positions have been checked by the first three strong classifiers, the program moves on to apply the rest of the strong classifiers (line 10 to 19 in Fig.9). The later part of the execution uses the same implementation as in the Loop-Changing scheme.

```
1: for all **WZ**                    // **WZ: window size**
2:   **Resize** image;
3:   **Integral** image;
4:   **Detect** {
5:     for all **WP**          // **WP: window position**
6:       for  **SC0:SC2**      // **SC: strong classifier** (cascade 0-2)
7:         for all **WC**      // **WC: weak classifier**
8:           If failed, label the position as negative; jump to the next WP;
9:
10:    for other **SC**        // **SC: strong classifier**
14:      for all **WP**        // **WP: window position**
15:        for all **WC**      // **WC: weak classifier**
16:          If failed, label the position as negative; jump to the next WP;
17:          // processor can better reuse the WC data stored in the cache
18:          // which significantly increases the data locality
19:    If passed all the SCs, label the position as positive;
20:  } // end of **Detect**
```

Figure 9. Pseudo code of the Classifier-Grouping scheme, which groups the execution of the first three (0~2) strong classifiers.

However, there are two main design concerns when applying the CG scheme on a parallel object detection algorithm. First, the CG scheme has imposed some overhead. During the execution of line 5 to 8 in Fig.9, CG scheme needs to record the positions which have passed each strong classifier. These positions will be read back when executing the subsequent strong classifiers. This will create extra memory accesses and data transactions.

Second, the number of bound classifiers could impact the overall performance. Binding too few strong classifiers cannot benefit the most from the data reuse. However, binding too many strong classifiers could possibly flush early loaded classifiers from the local cache and degrade the locality. Moreover, this number could change when the size of the local cache is different. To achieve the best performance, designers need to concern the induced overhead of the CG scheme and trade-off an appropriate number of bound classifiers.

This paper has successfully demonstrated the effectiveness of the CG scheme, and identified the application characteristics and key design concerns. Binding the first three strong classifiers is a decision from the empirical results. A more effective design trade-off requires a systematic design methodology, which is the next step in our future research.

VII. Conclusions and Future Work

The object detection enables a smart embedded device to recognize the surrounding environment and react properly. The intensive computation requirement requires a parallel object detection algorithm executing on a multi-core system. The memory bottleneck makes it a critical design concern to improve the data locality and take a full advantage of the on-chip cache. This paper analyzed the memory behavior of a parallel Viola-Jones algorithm, and proposed a Classifier-Grouping design scheme to enhance the data locality of the application. By running a multi-threaded object detection algorithm on a cycle-accurate multi-core simulator, the proposed approach can achieve up to 62% better performance when compared with the reference parallel design.

Our future work will focus on developing a systematic design methodology to perform the appropriate trade-offs on the critical design parameters, such as the extra overhead induced by the CG scheme, numbers of bound classifiers in CG scheme, cache sizes of processors, memory and interconnection latencies, etc.

ACKNOWLEDGEMENT

This work is sponsored by Nation Science Council under grant NSC99-2221-E-009-190.

REFERENCES

[1] H.W. Gellersen, A. Schmidt and M.Beigl, "Multi-Sensor Context-Awareness in Mobile Devices and Smart Artifacts," In Journal of Mobile Networks and Applications, Vol.7, Num.5, pp.341-351. 2002.

[2] R. Colin Johnson, "First MEMS gyro smartphone to ship in June," EETimes, May 2010. http://www.eetimes.com/electronics-news/4199279/MEMS-Gyro-Smartphone

[3] SixthSense Project, MIT Media Lab. http://www.pranavmistry.com/projects/sixthsense/

[4] Kinect for Microsoft Xbox. http://www.xbox.com/en-GB/kinect

[5] The TILE-Gx™ processor family processor. http://www.tilera.com/

[6] ARM cortex-A9 processor. http://www.arm.com/

[7] MIPS Technologies Announces Symmetric Multiprocessing (SMP) Support for Android™ Platform on MIPS-Based™ SoCs. http://www.mips.com/

[8] S. Leibson, "Memory is the Future Bottleneck in Multicore Servers," EDN News, March 2010.

[9] C. Zhang and Z. Y. Zhang, "A Survey of Recent Advances in Face Detection ", Microsoft Research, June 2010.

[10] P. Schaumont, B. C. Lai, W. Qin, I. Verbauwhede, "Cooperative Multithreading on Embedded Multiprocessor Architectures Enables Energy-Scalable Design," Proceeding 2005 Design Automation Conference (DAC), pp. 27-30, June 2005.

[11] M. Wolf and Monica S. Lam. A data locality optimizing algorithm. In ACM SIGPLAN symposium on Programming Languages Design and Implementation, pages 30–44, 1991.

[12] Cade, M.J.Cade and A. Qasem, Balancing Locality and Parallelism on Shared-cache Mulit-core Systems. 11th IEEE International Conference High Performance Computing and Communications, 2009. HPCC '09. Pages 188 – 195, June 2009

[13] M. Gaubatz, R.Ulichney, "Automatic red-eye detection and correction," in Proc. IEEE Int. Conf. Image Processing, vol. 1, pp. 804–807, 2002.

[14] M.J.Jones, D.Snow, "Pedestrian detection using boosted features over many frames," 19th International Conference on Pattern Recognition, pp.1-4, Dec.2008.

[15] T.T.Nguyen, H.Grabner, H.Bischof, B.Gruber, "On-line Boosting for Car Detection from Aerial Images", IEEE International Conference on Research, Innovation and Vision for the Future, pp.87-95, 2007

[16] T. Theocharides, N. Vijaykrishnam and M. J. Irwin, "A parallel architecture for hardware face detection", Symp on Emerging VLSI Technologies and Architectures, pp. 452-453, 2006.

[17] Y. K. Chen, W. L. Li and X.F. Tong, "Parallelization of AdaBoost algorithm on multi-core processors", IEEE SiPS 2008, Washington DC, 2008, pp.275-280.

[18] C.H.Chiang, C.H.Kao, G.R. Li, B.C. Lai, "Multi-Level Parallelism Analysis of Face Detection on a Shared Memory Multi-Core System," IEEE International Symposium on VLSI Design, Automation and Test, April 2011.

[19] Open Source Computer Vision, http://opencv.willowgarage.com/

[20] Y. Wei, X. Bing, C. Chareonsak, "FPGA Implementation of AdaBoost Algorithm for Detection of Face Biometrics", *In Proc. IEEE International Workshop Biomedical Circuits and Systems*, 2004.

[21] M. Yang, Y. Wu, J. Crenshaw, B. Augustine, R. Mareachen, "Face Detection for Automatic Exposure Control in Handheld Camera", *in Proc. IEEE International Conference Computer Vision Systems*, 2006.

[22] C. J. Gao and S. L. Lu, "Novel FPGA based Haar classifier face detection algorithm acceleration", *FPL 2008*, Heidelberg, September 2008, pp. 373-378.

[23] F. C. Crow, "Summed-Area Tables for Texture Mapping", *Computer Graphic*, vol. 18, no. 3, pp. 207-212, July 1984.

[24] P.Viola and M.Jones, "Rapid Object Detection Uisng a Boosted Cascade of Simple Feature," in Proc. CVPR, vol.1, pp.8-14, 2001.

[25] Shin-Kai Chen, Tay-Jyi Lin and Chih-Wei Liu, "Parallel Object Detection on Multicore Platform," in Proc. SiPs, 2009.

[26] D. Keppel, "Tools and Techniques for Building Fast Portable Threads Packages," UWCSE 93-05-06, U. Washington, 1993.

[27] C. Hammerschmidt, "HAVEit project proves series maturity of automatic driving," EETimes, June 2011.

[28] E.S Shin, V.J. Mooney, and G.F. Riley, "Round-robin Arbiter Design and Generation," 15th International Symposium on System Synthesis, pp.243-248, 2002.

[29] CAS latency, on Wikipedia, July 2011.