LETTER

DSP-Based Parallel Implementation of Speeded-Up Robust Features

Chao LIAO[†a)], ***Student Member***, **Guijin WANG**[†b)], ***Member***, **Quan MIAO**[†], ***Student Member***, **Zhiguo WANG**[†], ***Nonmember***, **Chenbo SHI**[†], ***Student Member***, ***and*** **Xinggang LIN**[†c)], ***Nonmember***

SUMMARY Robust local image features have become crucial components of many state-of-the-art computer vision algorithms. Due to limited hardware resources, computing local features on embedded system is not an easy task. In this paper, we propose an efficient parallel computing framework for speeded-up robust features with an orientation towards multi-DSP based embedded system. We optimize modules in SURF to better utilize the capability of DSP chips. We also design a compact data layout to adapt to the limited memory resource and to increase data access bandwidth. A data-driven barrier and workload balance schemes are presented to synchronize parallel working chips and reduce overall cost. The experiment shows our implementation achieves competitive time efficiency compared with related works.

key words: *digital signal processing, speeded-up robust features, SURF, parallel computing, image matching*

1. Introduction

Robust local image features are widely used in many state-of-the-art computer vision algorithms such as calibration, object recognition and motion tracking. As a sound representative, scale invariant feature transform (SIFT) is known for its distinctiveness against image scaling, rotation and view point change [1]. However, its excellent matching result is acquired at the cost of a heavy computation burden. This disadvantage is also common for other local features like maximally stable extremal region (MSER) [2], and limits their application to PC systems only. To relieve this burden, speeded-up robust features (SURF) is proposed as a light weighted alternate while maintaining a similar performance [3]. Meanwhile, modern DSP chips now acquire more strength with less energy consumption as the hardware techniques develop. There emerges a possibility to extract local features on embedded systems within reasonable time.

Sinha [4] and Cornelis [5] ported SIFT and SURF to graphics processing unit (GPU). GPU plus CPU obtain impressive processing speed, but they still require a PC system and more power supply. Special purpose hardwares on FPGA can provide a good balance between speed and energy, these works are reported in Ehsan's survey [6]. Ko [7] implemented SIFT on a BlackFin DSP inside a wireless camera network. Their solution is not very fast due to the complexity of SIFT and the low speed of DSP chip. Arth [8] implemented MSER on a single fixed-point DSP for an object recognition system, which is most similar to our work. However, we place more emphasis on improving the efficiency by computing in parallel with multiple DSP chips. Zhang [9] decomposed SURF modules into parts and computed with several threads on quad-core CPU. This work is also parallel and achieves real-time performance. But it is not very suitable for DSP systems due to platform discrepancies in memory usage and synchronization.

In this paper, we propose an efficient parallel computing framework for speeded-up robust features oriented towards multi-DSP based embedded system. With a major concern on latency, we optimize a few modules with high call-frequency to improve the efficiency. Due to limited internal memory, we also design a compact data layout to increase data access bandwidth. A data-driven barrier and workload balance schemes are presented to synchronize parallel working chips and reduce overall cost.

2. Parallel Framework

An efficient parallel design on embedded system considers both the possibility of parallelism in algorithm and the characteristic of hardware platform. As we mainly concern on the latency rather than throughput, we also improve the efficiency of selected sub-routines with high call frequency.

2.1 Algorithm Overview

SURF algorithm consists of five phases to obtain interest points, see Fig. 1 (b). It uses determinant of Hessian (DoH) detector (Eq. (1)) to locate blob-like structures in image. To achieve scale invariance, several scale space pyramids, referred to as octaves, are created by applying the detector on images filtered at different scales. An octave typically consists of 4 Hessian planes with their scale size specified in an increasing manner. SURF combines integrate image along with boxlet filters to approximate second order Gaussian derivatives. The approximation can be evaluated at a very low cost independent of filter size, hence speed up the generation of octaves.

$$H(x,y,\sigma) = \begin{bmatrix} \frac{\partial^2 G}{\partial x^2} & \frac{\partial^2 G}{\partial x \partial y} \\ \frac{\partial^2 G}{\partial y \partial x} & \frac{\partial^2 G}{\partial y^2} \end{bmatrix}, G_{x,y,\sigma} = \frac{e^{-(x^2+y^2)/2\sigma^2}}{2\pi\sigma^2} \qquad (1)$$

A non-maximum suppression is then performed in the

Manuscript received September 27, 2010.
Manuscript revised November 30, 2010.

[†]The authors are with the Department of Electronic Engineering, Tsinghua University, Beijing 100084, China
a) E-mail: liaoc02@mails.tsinghua.edu.cn
b) E-mail: wangguijin@tsinghua.edu.cn
c) E-mail: xglin@tsinghua.edu.cn

DOI: 10.1587/transinf.E94.D.930

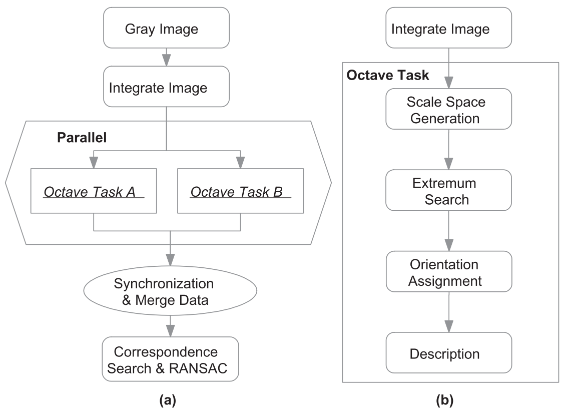

Fig. 1 Parallel computing of the octaves.

$3 \times 3 \times 3$ neighborhood inside an octave. The location where the DoH value is extreme has the potential to be an interest point. However, some further examinations like edge and contrast should be performed to reduce false alarm.

To synthesize descriptor, SURF first assigns a reproducible orientation. About 100 sample points are selected within the interest point neighborhood and applied with Haar wavelet filter along x, y directions. The dominant orientation is estimated from the distribution of their responses. After that, SURF constructs a square region centered at the interest point and aligned along the dominant orientation. Haar wavelet responses extracted within the square region are concatenated to produce a 64-dimensional descriptor.

2.2 Octave-Level Parallelism

We analyze the critical path of SURF to find out which parts can be computed in parallel. Critical path refers to the longest chain of dependent calculations. Every program fragment in the critical path depends on the result of prior fragment. Hence they must be executed in order and no program can run faster than its critical path. Bernstein's Conditions can help us to decide whether two fragments are independent. For two program fragments P_i, P_j, let I_i, I_j be their input, and O_i, O_j be their output. P_i is independent of P_j if they satisfy the conditions in Eq. (2).

$$I_j \cap O_i = \phi, O_j \cap I_i = \phi, O_i \cap O_j = \phi \tag{2}$$

Bernstein's Conditions clarifies that previously introduced 5 steps in computing an octave are all on critical path, see Fig. 1 (b). However, the octaves themselves are also proved to be independent to each other and hence can be computed in parallel. This conclusion also stands for other interest point detection algorithms such as SIFT and MSER.

With the knowledge from critical path, we propose the parallel SURF detection and matching framework in Fig. 1 (a). Octaves are computed on multiple DSP chips and their results are then merged together in synchronization before used to query against reference data for correspondence.

Computing octaves in parallel not only reduces the overall calculation time, but also saves the memory space. If all octaves require an allocation, the DSP memory would fail to meet their need. But since octaves are independent, we can only allocate one memory section for the largest octave, and other octaves can re-use its space.

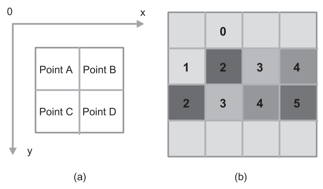

Fig. 2 Compute integrate image in parallel.

2.3 Integrate Image

SURF applies boxlet filter on integrate image to produce a fast approximation of Gaussian filter response. Here we propose a method to compute integrate image in parallel at instruction level. Image integration is usually carried out from top-left to bottom-right. The integration value is the sum of top and left pixels. In Fig. 2 (a), the execution order would be point A-B-C-D. But with Berstein's Conditions in Eq. (2), we notice that point B and C are independent, as their input are point A and self-pixel. Hence the critical path is point A-(BC)-D. Point B and C can be computed in parallel. The optimized procedure is performed line by line, with two rows calculated in one horizontal sweep, see Fig. 2 (b). The numbers indicate the execution order. Points with same numbers are computed in parallel with long instructions. This improvement saves more than 25% time compared with the regular method. But the overall gain on speed is not quite obvious since integration only takes a small portion in SURF.

2.4 Parallel Search

For image matching problems, the purpose of extracting interest points is to establish correspondences between two pictures. The correspondences, or say the matching points, are found by comparing the Euclidean distance of descriptors between one newly detected SURF point and the points stored in the reference database. As this search procedure is again independent for each query point, we can also carry it out in parallel. Each core can query its subset of detected points against the reference data. Their results are then combined to a formal correspondence set. This way not only brings better search speed, but also reduces the amount of data need to be transferred, since sometimes it is the correspondences rather than SURF points that are wanted.

3. DSP Implementation

The implementation of algorithm must always adapt to specific hardware. In our application, we use floating-point DSP chips TigerSHARC201 to evaluate our method.

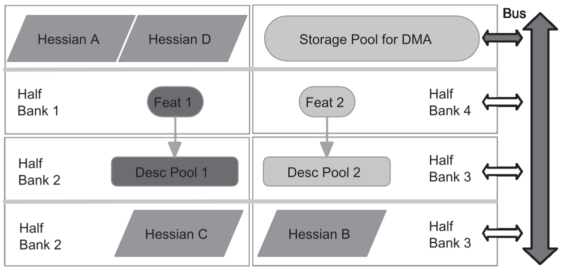

Fig. 3 Data layout in memory.

3.1 Data Layout

The TigerSHARC201 memory consists of 6 full banks which can be further divided into half-banks. Each half-bank is equipped with a read/write cache. With such a specification, neither the maximal consecutive data segment nor the heap size can exceed a full bank. Figure 3 illustrates our raw data-layout inside memory. Hessian A-D represent the Hessian matrices under 4 scales in one octave. They are split up due to the maximal volume constraint. While allocating them, we mainly consider reducing read/write conflicts with other contents in the same bank. e.g. The search extreme step triggers concurrent data access for Hessian ABC or BCD. By storing them in different half-banks, the bus can utilize 3 caches hence increase the access bandwidth. A D can stay in one bank as they are not conflicting. Basic SURF point information such as position, scale and orientation are stored in structure arrays Feat1 and Feat2, while their corresponding descriptors are stored in separated pools from other banks. Feat1 and Feat2 together form ping-pong buffers, which enable the algorithm to work in place with image sequences. The storage pool for communication data locates in an isolate half-bank, which reduces the conflict between DSP core and communication device.

3.2 Workload Balance

While assigning tasks to different cores, it is better that they can get finished in almost the same time. Otherwise one core would have to wait for the others to finish before entering next phase. The boxlet method in SURF simplifies Gaussian filtering into fixed number of summations, so the workload for each octave can be roughly estimated by its hessian matrix size. Since the matrix width and height halves as the octave index increase, the workload for 4 octaves are 1, 1/4, 1/16 and 1/64. Octave0 is most expensive and costs even more than the sum of the rest. For dual-core platform, it is reasonable to assign octave0 to one core, and the rest to the other. This scheme cannot cover the gap completely, but is easy to implement. Another possible scheme would be to divide octave0 into halves and assign them to different cores. This scheme requires additional data merging step and is more suitable for platforms with more than 3 cores.

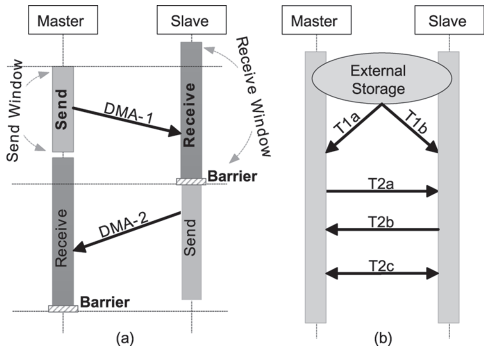

Fig. 4 Data exchange and synchronization with DMA.

3.3 Synchronization

Synchronization is the check point where parallel working units exchange data and coordinate their progress. Synchronization guarantees correct execution order and data integrity, although it also brings complicated organism and noticeable communication overhead. For shared-memory system, it is usually implemented in a centralized manner. Each processor updates a state variable stored in shared memory to indicate its arrival to certain check point. Then they poll these state variables to determine others' arrival. Once all true, they are permitted to continue. Otherwise, they are all stalled as if these state variables formed a barrier.

However, high speed shared memory is rarely provided on multi-DSP platform and centralized state indicators are not available. To overcome this difficulty, we present a data-driven barrier to replace the centralized one. While a centralized barrier is visible to all processors, a data-driven barrier is only visible to local processor and to ensure local program halted till its input data arrive. The key observation is that all what a program fragment need to operate is proper input data. In Fig. 4 (a), each core has a data-driven barriers set at the bottom of receive window. Local barriers may take place with slight time difference, but they together also keep the time order that data exchange completes before the program goes to next phase.

Another characteristic is that shared data are maintained by communication. Each DSP has different memory space and keeps its own copy of shared data. This is different from shared memory system where only one copy would be sufficient for all threads. e.g., as all cores need to share detected points, master and slave cores will duplicate their detection result and transfer to each other through physical channels to complete a data exchange/sharing.

Time order is the most concerned problem in communication. Data transfer is implemented with direct memory access (DMA) technique which requires receive time windows open earlier and close later than send windows, see Fig. 4 (a). This requirement is very challenging if many transfers are involved. Unfortunately, there are as many as 5 DMA transfers in one process cycle, see Fig. 4 (b). T1a and T1b load image and reference data from external storage into internal memory. T2a sends integrate image from

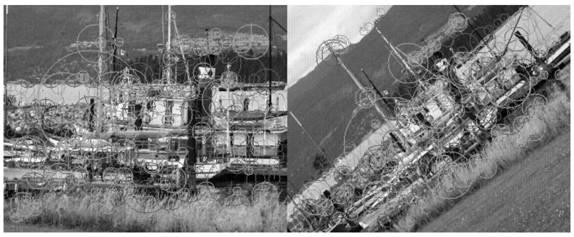

Fig. 5 Detected SURF points in Boat images.

master to slave core. After that both cores start detection in parallel. Suppose master would finish first, it waits for T2b to send back matching points from slave core. In the end, both cores exchange their interest points by T2c. The time windows for these DMA must be carefully designed. However, by using more DMA channels this problem can be alleviated as the interlacing among these DMA are weakened.

4. Experiment

We test the proposed method on an evaluation board of 2 TigerSHARC DSP chips. Each chip has 600 MHz main frequency, and 3 MB internal memory. Due to the resolution restriction from external camera, the input image size is constrained to 320×256. The performance is compared with the CPU implementation of Bay [3] on the benchmark data set provided by Mikolajczyk [10]. Both produce identical matching results while the detected point numbers have a little difference, which is caused by different threshold setting in extreme search. Figure 5 shows our results on Boat images.

We compare the time efficiency with other works in Table 1. The original SURF by Bay [3] takes about 100 ms on PC with Pentium IV 3.0 GHz CPU. Our method achieves a similar detection time of 120 ms, but with much slower chips (600 MHz×2) and limited memory resources. Compared with other interest point detection algorithms based on DSP, our result is also time efficient. Ko [7] implemented SIFT on a Black Fin chip with a processing time longer than 3 second. Part of the reason for the long time is that SIFT is much more complicated than SURF. Arth [8] implemented MSER on one single fixed-point chip with a processing time of 257 ms. As fixed-point chip is usually faster than floating-point chip, our method is also considered competitive. From parallel computing perspective, our special advantage is that we allow for even more DSP cores. Extra time efficiency can be achieved by further dividing octaves into smaller pieces and calculate them with additional cores.

To better investigate the effect of our parallel framework, we compare its time cost with serial settings. Figure 6 illustrates the time change for same image contexts. The serial one takes more than 180 ms, while by calculating in parallel, the time cost is reduced to 120 ms. A gain of about 60 ms (33%) is achieved. The speed is not doubled mainly due to the waiting cost in synchronization caused by unbalanced workloads. The overhead from memory communication is relatively small. As core-to-core link port DMA speed is about 0.5 MB/ms, transmitting the largest data section (integrate image, 320 kB) costs less than 0.64 ms.

Table 1 Performance comparison. The first row is with CPU.

Author	Time	Size	Algorithm	Platform
Bay [3]	100 ms	320×256	SURF	PentiumIV 3.0 GHz
This	120 ms	320×256	SURF	TigerSHARC201×2
Arth [8]	257 ms	352×288	MSER	TMS320C6414
Ko [7]	>3 s	320×256	SIFT	BlackFin

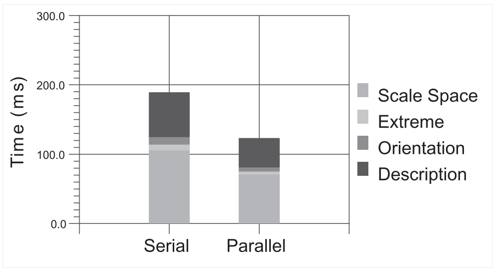

Fig. 6 Time cost proportion for serial&parallel settings (dual core).

5. Conclusion

In this paper, we present a fast parallel implementation of SURF on DSP-based embedded systems. We investigate octave-level parallelism and study the key problems of memory layout, workload balance and synchronization. We achieve competitive time efficiency with 2 TigerSHARC chips compared with related works. In the future, we will further explore the possibility to incorporate more chips and reduce the overhead in multi-core synchronization.

References

[1] D.G. Lowe, "Distinctive image features from scale-invariant keypoints," Int. J. Comput. Vis., vol.60, no.2, pp.91–110, 2004.

[2] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide-baseline stereo from maximally stable extremal regions," Image Vis. Comput., vol.22, no.10, pp.761–767, 2004.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," CVIU, vol.110, no.3, pp.346–359, 2008.

[4] S.N. Sinha, J.M. Frahm, M. Pollefeys, and Y. Genc, "Feature tracking and matching in video using programmable graphics hardware," Machine Vision and Applications, pp.1–11, 2007.

[5] N. Cornelis and L. Van Gool, "Fast scale invariant feature detection and matching on programmable graphics hardware," CVPR Workshops, pp.1–8, 2008.

[6] S. Ehsan, A.F. Clark, et al., "Hardware based scale and rotation invariant feature extraction: A retrospective analysis and future directions," Proc. ICCEE, pp.620–624, 2009.

[7] T. Ko, Z.M. Charbiwala, et al., "Exploring tradeoffs in accuracy, energy and latency of scale invariant feature transform in wireless camera networks," ICDSC, pp.313–320, 2007.

[8] C. Arth, et al., "Using robust local features on DSP-based embedded systems," Embedded Computer Vision, pp.79–100, 2009.

[9] N. Zhang, "Computing optimized parallel speeded-up robust features (P-SURF) on multi-core processors," Int. J. Parallel Programm., vol.38, no.2, pp.138–158, 2010

[10] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors." IEEE Trans. Pattern Anal. Mach. Intell., vol.27, no.10, pp.1615–1630, 2005.