J. Parallel Distrib. Comput. 92 (2016) 1–14

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Parallelizing image feature extraction algorithms on multi-core platforms

Yunping Lu [d,b], Yi Li [a,b,c], Bo Song [a,b,c], Weihua Zhang [a,b,c,*], Haibo Chen [e], Lu Peng [f]

[a] Software School, Fudan University, Shanghai, China
[b] Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China
[c] Parallel Processing Institute, Fudan University, Shanghai, China
[d] School of Computer Science, Fudan University, Shanghai, China
[e] Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University, Shanghai, China
[f] Division of Electrical and Computer Engineering, Louisiana State University, United States

HIGHLIGHTS

- Analysis and evaluation of various parallelism in image feature extraction algorithms.
- Observations on parallelism constraints in image feature extraction algorithms.
- An efficient adaptive pipeline scheme with good scalability.
- A power-efficient parallelism algorithm for various workloads.

ARTICLE INFO

Article history:
Received 11 October 2014
Received in revised form
10 August 2015
Accepted 2 March 2016
Available online 10 March 2016

Keywords:
Image feature extraction
SIFT
SURF
Adaptive pipeline
Multi-core

ABSTRACT

Currently, multimedia data has become one of the most important data types processed and transferred over the Internet. To extract useful information from a huge amount of such data, SIFT and SURF, as two most popular image feature extraction algorithms, have been widely used in many applications running on multi-core platforms. However, limited parallelism in existing designs makes it hard or impossible to apply them in many applications with real-time requirements. Therefore, it has become one of the major challenges to improve the processing speed of image feature extraction algorithms.

In this paper, we first analyze the parallelism constraints in the algorithms, such as imbalanced workloads and indeterminate time distributions. Based on such analyses, we present an adaptive pipeline parallel scheme (AD-PIPE) to adjust the thread number in different stages according to their workloads dynamically, which achieves a balanced partition for constant input workloads. Furthermore, we also implement a power efficient version (AE-PIPE) for AD-PIPE through scheduling threads based on variable input workloads. Experimental results show that AD-PIPE achieves a speedup of 16.88X and 20.33X respectively over SIFT and SURF on a 16-core machine. Moreover, AE-PIPE achieves up to 52.94% and 58.82% power saving with only 3% performance loss.

1. Introduction

As our society has entered a data-centric world, a huge amount of data is transferred and processed over the Internet. As indicated in the forecast of CISCO Inc. [8], until 2014, the data quantity generated every month will reach more than 0.6 million PB. Among such a huge amount of data, multimedia data has become one of the most common data types being processed.

With the dramatic increase of multimedia data, it is vitally important to continually collect, index and retrieve such ever-increasing data to extract useful information and understand them. Therefore, image feature extraction algorithms (IFEAs), as the fundamental components of image and video retrieval applications, have been designed and developed for many years. Among the IFEAs, SIFT (*Scale Invariant Feature Transform*) [20,21]

* Corresponding author at: Software School, Fudan University, Shanghai, China.
E-mail addresses: luyping@sina.com (Y. Lu), yee.lie@gmail.com (Y. Li), espiesong@gmail.com (B. Song), zhangweihua@fudan.edu.cn, whzhang.fd@gmail.com (W. Zhang), haibochen@sjtu.edu.cn (H. Chen), lpeng@lsu.edu (L. Peng).

http://dx.doi.org/10.1016/j.jpdc.2016.03.001
0743-7315/

and SURF (*Speeded Up Robust Features*) [4] are two most popular ones [25,2]. They have been widely used in many applications, such as image and video retrieval [18,27], object recognition [15,3], and face recognition and authentication [7,10].

However, the processing speed of the current image feature extraction algorithms (IFEAs) still has a large room to be improved. For example, SIFT can only process about 1.8 images or frames per second on an average CPU and SURF can only process about 2.6 images or frames per second based on our experiments. Such results make them impossible to be used in scenarios with real-time requirements, such as large-scale content-based image retrieval or object recognition systems. The major reason stems from their design complexity: to make the algorithms insensitive to rotation, scaling, contrast and viewpoint changing, some complex transformations are included in the design of the IFEAs. Moreover, to guarantee retrieval accuracy, hundreds or thousands of feature points are extracted to represent an image or a frame. Each feature point will be further described with information around it and filled into a high-dimensional vector. Therefore, the algorithms are not only computation intensive but also data intensive.

The popularity of multi-core architecture and the increase of computation resources on such platforms provide a new opportunity to accelerate the processing speed of the IFEAs. Such opportunity has been evidenced by recent efforts on parallelizing these algorithms, such as [14,35,36]. However, the achieved speedup can still be further improved. For example, the parallelizing effort in [37,29] only achieves a speedup of about 6X on a 16-core machine.

In this paper, we first systematically analyze the characteristics related to parallelization of the IFEAs (SIFT and SURF). We find that there exist some inherent parallel constraints, such as imbalanced workloads in different feature points, sub-blocks, images, and indeterminate time distributions of different functions. To alleviate these constraints, we design and implement an adaptive pipeline parallelism scheme (AD-PIPE) for the IFEAs with constant input workloads. AD-PIPE partitions different functions into different pipeline stages in a producer–consumer manner. Such a design can efficiently overcome the constraints of imbalanced workloads. To alleviate the constraint of indeterminate time distributions among stages, we apply an adaptive strategy. The strategy can dynamically check workloads in different pipeline stages and adjust the thread number in different stages to achieve a balanced partition. To further improve AD-PIPE from the perspective of both performance and energy, we also extend AD-PIPE to design and implement a power-efficient version for variable input workloads, called AE-PIPE. AE-PIPE can adjust the number of threads between the working state and the idle state based on input workloads to reduce unnecessary computation resources.

Experimental results show that such designs are efficient and scalable. AD-PIPE can achieve a speedup of 16.88X and 20.33X respectively over SIFT and SURF on a 16-core commodity machine and a real-time processing speed of about 30 and 52 images or frames per second. For AE-PIPE, it can achieve about 23.81% and 25.30% power saving respectively over SIFT and SURF for variable input workloads, and up to 52.94% and 58.82% power saving for constant workloads with only 3% performance loss.

In summary, the main contributions of this paper can be summarized as follows.

- An analysis and an evaluation of various parallelism in the IFEAs, including image level, block level, scale level, pipeline level and their combinations.
- The observations on parallelism constraints in the IFEAs, including the imbalanced workloads and indeterminate time distributions.
- The design and implementation of an efficient adaptive pipeline scheme for the IFEAs with constant input workloads, which is suitable for the IFEAs and outperforms prior designs with good scalability.
- The design and implementation of a power-efficient parallelism for the IFEAs with variable input workloads, which can schedule threads based on input workloads.

The paper is organized as follows. Section 2 gives an overview of various IFEAs and related acceleration work. Section 3 describes SIFT and SURF in brief and presents a systematic characteristics analysis of them. Section 4 focuses on the design and implementation of our adaptive pipeline parallel scheme. Adaptive power-efficient pipeline parallelism is discussed in Section 5. In Section 6, we show detailed evaluation results on multi-core platforms. Finally, we conclude our work in Section 7.

2. Related work

In this section, we first introduce some image feature extraction algorithms (IFEAs). Then, we discuss some previous acceleration efforts on the IFEAs.

2.1. Image retrieval algorithms

As the fundamental components of image and video retrieval applications, image retrieval algorithms extract features to represent an image (or a video frame). Image features usually can be divided into two categories: global feature-based and local feature-based.

Global feature-based algorithms (GFAs): GFAs usually use a single feature to represent an image or a video frame, such as color histogram and texture. Due to using only one feature to represent the entire image, GFAs fast extract the features but with low precision in matching (more than 30% error rate [28]), which limits their applications. Furthermore, the algorithms cannot be used to retrieval images after some transformations, such as re-sizing and cropping [5]. Thus in many applications, GFAs are insufficient.

Local feature-based algorithms: In contrast to GFAs, local feature-based algorithms (LFAs) extract hundreds or thousands of features to represent an image. To guarantee the retrieval accuracy, the algorithms generally include some complex computation steps. Due to high precision in matching, they have been widely used in many real-world applications.

Among all LFAs, SIFT and SURF are two most popular and robust ones [25,2]. Thus, we will mainly focus on them in this paper. SIFT [20,21] is the most publicly accepted and robust local feature-based image extraction algorithm. To meet different requirements, there are many variants of SIFT, such as GLOH [25], and PCA-SIFT [17]. Since all the variants are based on SIFT, in this paper, we only focus on the original SIFT algorithm for research. Another widely-used algorithm is SURF [4]. After proposed in 2006, it has been applied to many applications, and tends to be an efficient alternative of SIFT. Both of two algorithms are insensitive to various transformations, such as scaling, rotation and illumination.

2.2. Previous acceleration efforts

Parallelization, as an efficient approach, has been widely used in different areas for performance acceleration, such as atmosphere prediction [32,31,19], graph processing [34], and architecture simulation [30]. Since the LFAs include complex computations and have to describe hundreds of feature points, they are time-consuming, which limits their application fields in the real world with real-time requirements. In order to solve the problem, many efforts have been done to accelerate SIFT and SURF through exploiting inherent parallelism in them.

To understand the characteristics of these LFAs better, Lu et al. [23] constructed a multimedia retrieval benchmark and compare their characteristics with traditional multimedia applications. Zhang et al. [22] analyze the gaps between current hardwares and the architectural characteristics of multimedia retrieval applications and proposed a novel hardware architecture for multimedia retrieval applications.

Zhang et al. [37] implemented a parallel SIFT algorithm and showed a 6.4X speedup on an 8-core machine. Feng et al. [14] achieved a speedup of 11X on a 16-core machine. Both of them exploited block level parallelism, which parallelized SIFT through partitioning an image into different sub-blocks. Although two designs achieve some performance improvements, both their results include around 2X speedup from serial optimizations such as cache optimization. Therefore, the true speedup gained from the parallelism was about 3X for an 8-core machine and 5.5X for a 16-core machine. In [35], Zhang implemented a scale level parallelism over SURF on multi-core CPU. In order to overcome imbalanced workloads of scale level, they improved their design with block level parallelism in [36]. As evaluated in [13], their design actually gained a speedup of 6.0X from its parallelism for a 16-core machine.

From previous works, it is obvious that large gap exists between parallel designs and the ideal speedup, which means that there might be some limitations in those designs or in SIFT and SURF themselves. As analyzed in the following sections, there exist some constraints, such as imbalanced workloads. Such limitations greatly affect the performance of prior parallel designs. Researches mainly focused on mapping a LFA onto some specific hardware by exploiting a specific form of parallelism, instead of analyzing parallel characteristics of the LFAs in detail and designing a real effective parallel scheme. Moreover, most of them don't consider power efficiency in their designs, which is one of most important issues in real-world environments. Therefore, in this paper, we first comprehensively analyze parallel constraints. Based on the analysis, we design an effective parallel framework for the LFAs. Our approach can achieve not only good performance but also power efficiency.

3. SIFT and SURF overview

In this section, we first briefly introduce SIFT and SURF, and then analyze their characteristics, such as the major parallelism in them. At last, we analyze the factors which will influence the efficiency of different parallelizations.

3.1. SIFT and SURF

SIFT and SURF are two most popular and robust LFAs [25,2]. As shown in Fig. 1, both of them consist of an initialization stage, feature detection stage, and feature description stage. An overview of their workflow is shown as follows:

- **Initialization stage**: This stage does some initialization work, including loading the image, getting the intensity of each image pixel and calculating the intermediate representation.
- **Feature detection**: This stage is to detect *feature points* (also called *points*) in an image or a video frame. As shown in Fig. 2, a $m * n$ scale space pyramid is constructed to guarantee scale invariance. SIFT and SURF use different computation methods to construct the pyramid. In SIFT, image size is varied between adjacent octaves and the Gaussian filter is repeatedly applied to smooth subsequent intervals in the same octave. After that, SIFT constructs difference-of-Gaussian pyramid by subtracting adjacent Gaussian intervals. SURF leaves the original image unchanged and varies only the filter size to construct the pyramid, which is more computationally efficient than SIFT. After building the scale space, each point in the pyramid is compared with its surrounding 26 points in a $3 * 3 * 3$ cube. If its value is the extreme value, the point is extracted as a feature point candidate. To guarantee the quality of extracted points, the candidates with low contrast or localized along edges are discarded. The remaining candidates are the final feature points.
- **Feature description**: In this stage, each detected point will be described by an n-dimensional vector. First, an orientation value is calculated based on the information around it to make the algorithm rotation invariant. Then, a descriptor window is constructed, and the feature vector is computed based on the orientation information. The value of the feature vector is also normalized to keep the algorithm illumination invariant. In SURF, each feature point is described as a 64-dimensional vector. And in SIFT, a 128-dimensional vector is used.

3.2. Available parallelism

There exist several levels of parallelism in SIFT and SURF, including image level, scale level, block level, and pipeline as shown in Fig. 3. They are described as follows:

- **Image level**: On image level, images are divided into several groups and the groups can process LFAs in parallel. There are two partition methods for image level parallelism. The first one is a static partition, which groups the images statically. The other is a dynamical manner. After each thread finishes its current work, it gets a new one from unprocessed images.
- **Scale level**: Each scale in the scale-space pyramid is constructed and detects feature points independently. Thus scales can be processed concurrently.
- **Block level**: Each input image can be divided evenly into some sub-blocks, and blocks can be processed by LFAs concurrently.
- **Pipeline**: Different stages in SIFT and SURF, such as detection (Det) and description (Des), can be distributed to different cores and work in a pipeline manner. Input images are treated as data streams to flow through all stages one by one.
- **Their combination**: The parallelism stated above can also be combined together. For example, the combination of image level and block level first divides the images into groups, and then each image in the group can use the block level parallelism to further parallelize the algorithm.

3.3. Constraints for parallelization

Before we get down to implement efficient parallel LFAs, it is better to understand their characteristics. Based on the characteristics analysis, we can find what and where the limitations exist. Only by overcoming these obstacles can we design and implement a more efficient parallel algorithm. Hence, we analyze the major characteristics that may be potential constraints for parallelizing SIFT and SURF. For simplicity, we mainly use two image sizes for later evaluation. The image set of $640 * 480$ as standard or small image set, and $1600 * 1200$ images are chosen as the large image set.

3.3.1. Indeterminate time distributions

We investigate time distributions of different processing stages in LFAs for various image sizes. Here, to illustrate the problem more clearly, we choose two coarse-grained stages, detection and description, as an example. We downloaded 20 $2048 * 1536$ images from the Internet and converted them to various image sizes. Fig. 4 gives the time ratio of two stages (description/detection) for different image sizes. The value is the average results of 20 images with the same size. The results show that time distributions

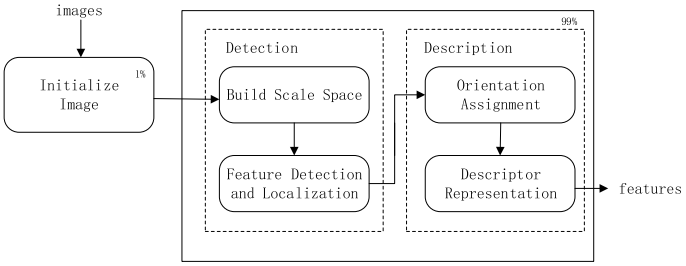


Fig. 1. Workflow of SIFT and SURF.

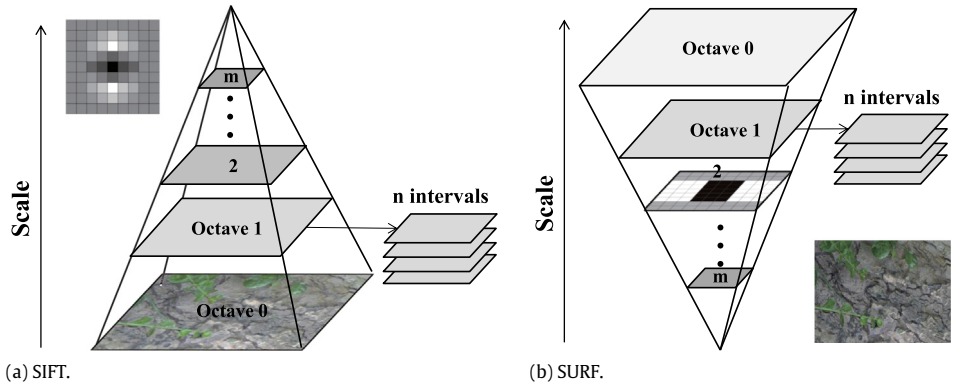


Fig. 2. The pyramid structure in SIFT and SURF. The pyramid consists of *m* octaves; each octave consists of *n* intervals.

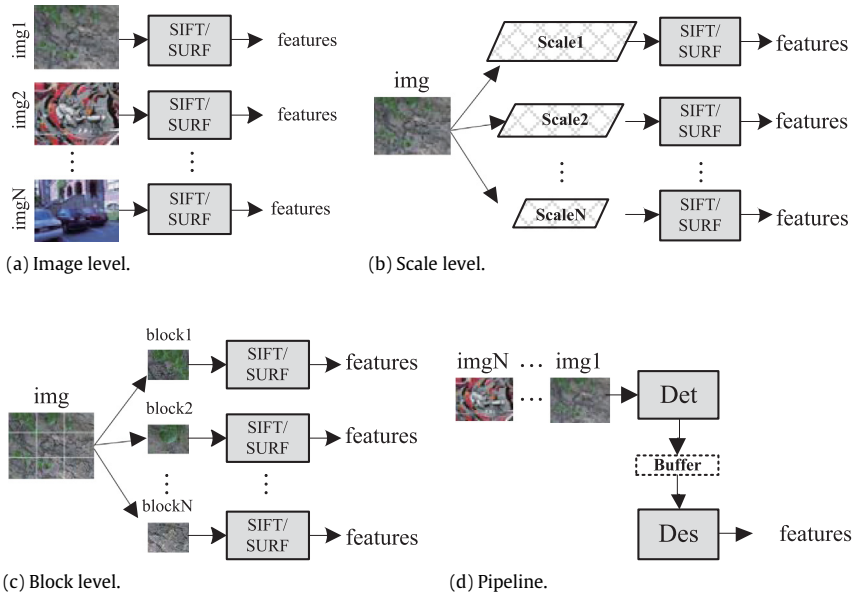


Fig. 3. Different primary parallelism in the LFAs (SIFT and SURF). Blocks with shadow are those components mapped on different cores.

of the two stages are changeable for different image sizes. The major reason is the difference in their image sizes and the number of points. The larger the image is, the more time detection stage consumes. The more the points are, the more time description stage consumes. On average, when the size of the image is up-scaling, the ratio decreases. Actually, time distributions for the images with the same size are also various due to their various feature point number.

3.3.2. Imbalanced workloads

Another big constraint of the LFAs is imbalance workloads. The imbalance exists on image level, scale level, block level, and feature point level.

Imbalance on image level: Fig. 5 shows the number of feature points detected in each image in the small image set. As the results shown, some images have a larger number of feature points while others have a smaller number of feature points. The various

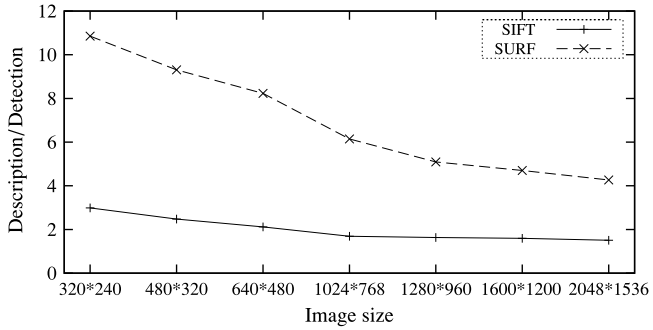


Fig. 4. Time distributions between detection and description stages. The ratio is the time of description stage dividing that of detection stage.

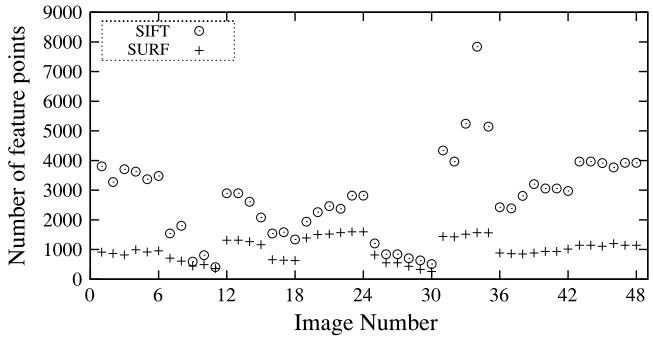


Fig. 5. Imbalance on image level.

number of feature points in the images leads to different execution time, which leads to imbalanced workloads in image level.

Imbalance on scale level: Furthermore, during building the scale pyramid, workloads of each scale are also imbalanced. Fig. 6 illustrates such a condition, where the processing time of each scale for building pyramids is shown. We divided the diagram of pyramid structure into rows where each row stands for each octave. The execution time of each octave is labeled in the right corner of the row, and that of each interval is labeled in the corresponding left parallelograms. The diagram illustrates that workloads for each scale are imbalanced. The upper scales need more time to be processed.

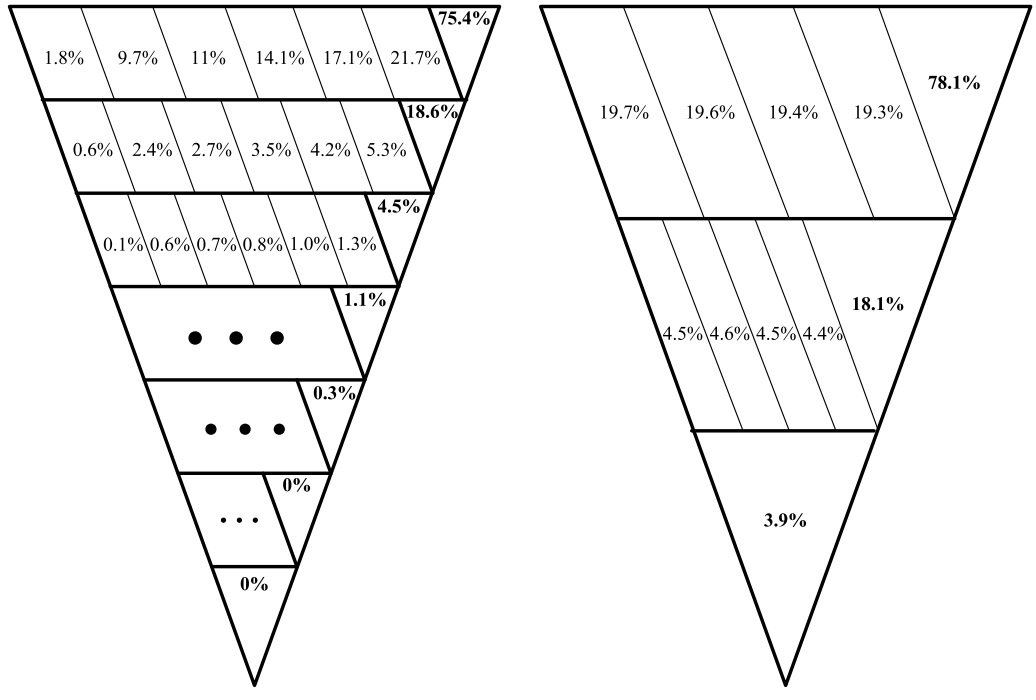
Imbalance on block level: Imbalance workloads also exist on block level. To illustrate this problem, we evenly divide a $640 * 480$ image into $4 * 4$ blocks and collect the number of feature points detected in each block. The data are shown in Fig. 7. As the data shown, the number of feature points detected in each block is different. Therefore, the parallelism in prior research [37,14,36] cannot achieve a good performance because of imbalanced workloads on block level.

Imbalance on point level: Workloads of different points are also imbalanced. Fig. 8 shows the description time of each feature point of SIFT and SURF in a standard image. The major reason of such a result is that when a feature point is in an upper scale, the diameter for orientation computation is larger than that of the feature point on a lower level, which will lead to different computation cost for different points in different locations. Therefore, it cannot achieve a balanced partition to exploit the parallelism on feature point level.

3.3.3. Constraints for parallelization

Balanced workloads have been one of the most important issues to achieve good performance and good scalability for parallel applications. Although there are several types of parallelism in SIFT and SURF, the characteristics, such as indeterminate time distributions and imbalanced workloads, greatly affect their efficiency.

- **Constraints for image level parallelism**: The major constraint is imbalanced workloads in image level parallelism since some images may have thousands of points while others may have little. Therefore, it is impossible to achieve a balanced partition when allocating images to different threads.
- **Constraints for block or scale level parallelism**: The performance of block level or scale level parallelism is also limited by



(a) SIFT. (b) SURF.

Fig. 6. Imbalance on scale level.

imbalanced workloads in them respectively. Moreover, to exploit scale level or a block parallelism, synchronization cannot be avoided. Taking block level parallelism of SIFT as an example, synchronization has to be involved after each scale of Gaussian pyramid is built because the calculation of the Gaussian scale in one block may use the data of other blocks. Thus, frequent synchronization is another factor that affects the performance.

- **Constraints for pipeline parallelism**: It appears that pipeline parallelism will not be influenced by the constraint of imbalance workloads since each point detected in the former stage is transferred equally to the latter ones. However, the indeterminate time distributions make the design of static pipelines impractical. Moreover, it is difficult to achieve a scalable and balanced partition when the core number increases.

4. Scheduling for constant workloads

In a multimedia retrieval system, there generally are two input conditions. The first condition is constant input flow, which mean the incoming image or video frame number is a constant value. For example, the input images are read from a data set stored on computer disks. The second one is variable input flow, such as website user requests. Our goal is to overcome the parallel limitations for such inputs and fully utilize hardware resources. In this section, we design and implement an adaptive pipeline parallel scheme (AD-PIPE) for constant input flow, which is suitable for both SIFT and SURF and is more efficient and scalable compared to prior designs. We will first give out the theoretical analysis and discuss the basic design of AD-PIPE. Then we show the throttling mechanism to avoid thrashing threads between the two pipeline stages.

4.1. Design consideration

As discussed in Section 3.3, there exist some constraints when parallelizing the IFEAs. Such constraints will waste computation resources on multi-core platforms since they would involve a lot of waiting time, which will lead to poor performance. So we firstly analyze the utilization of parallel resources and illustrate our design consideration.

Suppose there is a set of cores $C = \{c_0, c_1, \ldots, c_{|C|-1}\}$ ($|C|$ denotes the number of elements in set C, and that is the number of cores here). When a parallel application is executed on cores in C, there are two kinds of time for each core c_i,: the working time T_{W_i} and the waiting time T_{I_i}. T_{W_i} stands for the actual working time to deal with real workloads. T_{I_i} represents the idle time involved by the parallel constraints in the IFEAs, such as imbalanced workloads. Therefore, when parallelized the IFEAs are executed, the utilization of each core is the proportion of its working time in its total execution time. The total utilization (U) of cores is the sum of these cores' utilization. It can be represented as follows:

$$U = \sum_{i=0}^{|C|-1} \frac{T_{W_i}}{T_{W_i} + T_{I_i}}.$$

Based on this formula, higher utilization means better performance. Thus, when parallelizing the IFEAs to multi-core platforms, we should maximize the utilization to achieve the best performance. It can be formulized as follows:

$$Maximize(U) \rightarrow Maximize\left(\sigma = \frac{U}{|C|}\right) \rightarrow 1$$

where σ denotes utilization rate of corresponding cores C.

To achieve this goal, we involve the thought of the greedy algorithm in our design. In other words, to get a globally optimal solution, we will try to make a locally optimal choice for each core.

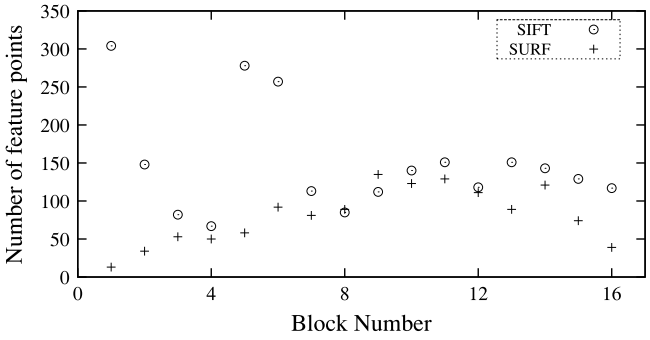


Fig. 7. Imbalance on block level.

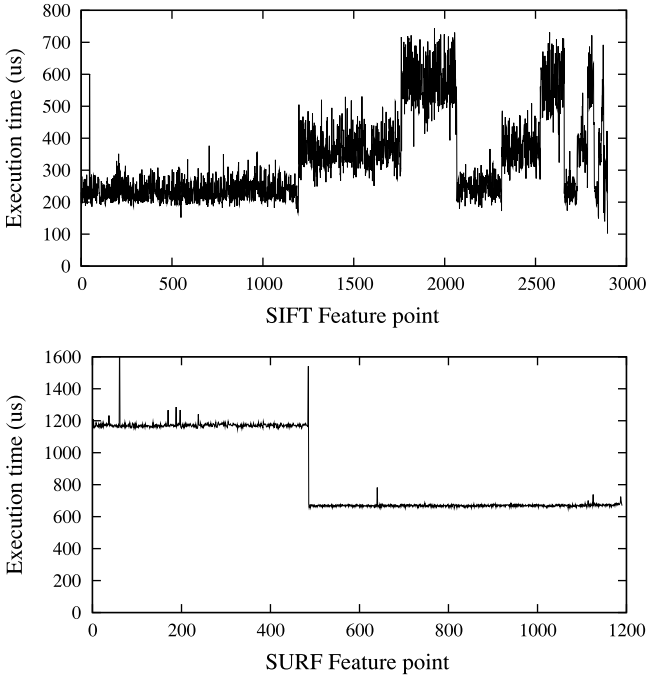


Fig. 8. Imbalance on feature point level.

Therefore, our goal can be converted to achieve the local optimal utilization u_i for each core c_i:

$$Maximize\left(u_i = \frac{T_{W_i}}{T_{W_i} + T_{I_i}}\right) \rightarrow 1.$$

Based on above analysis, we design and implement an adaptive pipeline parallel scheme (AD-PIPE) to alleviate the parallel constraints and maximize U when parallelizing the IFEAs. We mainly exploit the pipeline parallelism in our design, meanwhile eliminate the constraint of indeterminate time distributions with our adaptive strategy, which can maximize T_W. The basic design consideration is that the constraint of imbalanced workloads existing in other parallelisms is more difficult to be detected and adjusted since it is impossible to predict when and where imbalance workloads exist.

In a pipeline parallelism, a whole process of an application is partitioned into different stages, which works in a streaming manner. Input data flows through stages and are processed one by one. Therefore, there exist two major issues in the design of pipeline parallelism. The first issue is how to achieve a balanced partition, which can avoid unnecessary wait among pipeline stages. Generally, pipeline stages are statically partitioned based on the percentage of their execution time. However, the design of static pipeline

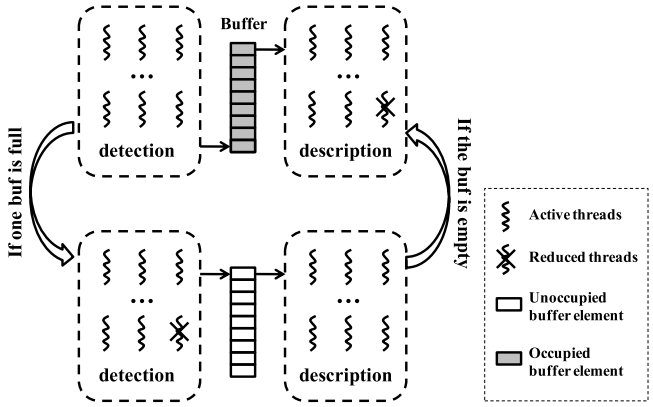


Fig. 9. The basic architecture of AD-PIPE.

has the limitation of indeterminate time distributions among different stages in SIFT and SURF for different inputs. The other issue is the efficiency of the communication between adjacent pipeline stages, which also has a great influence on the pipeline performance.

4.2. Adaptive pipeline design

To efficiently eliminate the constraint of indeterminate time distributions, we design an adaptive pipeline parallel scheme (AD-PIPE) over SIFT and SURF. In AD-PIPE, the number of threads of different stages is dynamically adjusted based on current workloads. If workloads are too heavy for one pipeline stage, its thread number will be increased, and the stages with low workloads will be decreased. However, when the number of pipeline stages increases, more synchronization between the adjacent stages will be involved, and T_ls for two stages will increase. To avoid this problem, we apply a coarse-grained pipeline partition. In other words, we only partition the SIFT and SURF into two stages: detection and description. As shown in Fig. 9, all threads are divided into two groups in AD-PIPE based on an initial proportion parameter, such as 1:3 or 1:1. When more computation resources are available, the threads for two stages will be increased based on the initial proportion parameter. Therefore, instead of designing a single-producer and single-consumer pipeline in prior research, we apply a multi-producer and multi-consumer pipeline. In such a design, some threads are doing detection and the others are doing description. Detection threads build scale-space pyramid, detect feature points concurrently and write them into the shared buffer, while description threads read feature points from the shared buffer and describe them concurrently. To detect whether there are imbalance workloads between two stages, we check the buffer states cyclically. When the buffer keeps full for a certain time, it means that detection stages run much faster than the description stages. In such a condition, the idle time for the threads in the detection stages will increase. To improve the utilization, the number of detection threads should be reduced, and that of description threads should be increased correspondingly. Similarly, when the buffer keeps empty for a certain time, it means that the description stages run much faster now. Therefore, some description threads should be changed back to do the detection work.

4.3. Buffer design

Besides the design of pipeline partition, buffer design must be carefully considered with this adaptive strategy for dynamical adjustment of thread numbers. For example, if we use 1:3 static pipeline on a 16 core machine, that is, 4 detection threads and 12 description threads, we can use 4 local-shared single-producer and multiple-consumer buffers, one of which is shared by a group of one detection thread and three description threads, to alleviate the buffer contention. However, this kind of buffer design is inefficient for our adaptive pipeline scheme. When the number of detection thread is decreased, idle description threads should be reassigned to other groups to help their description work. In a similar way, when the number of detection threads in a certain group is increased, description threads in other groups should be reassigned to help description work in this group. Therefore, this kind of reassignment is inefficient because workloads of each group may be imbalanced. Thus, in our AD-PIPE, we adopt a global-shared multiple-producer and multiple-consumer buffer to store all detected feature points from different detection threads. This global buffer is a circular queue, where detection threads store features to the tail of this global queue and description threads reads them from the head. We use semaphores to detect whether the buffer is full or empty and locks to assure atomic access to a buffer entry. To avoid unnecessary communication and reduce synchronization overhead, the feature points detected in an image are not partitioned and actually only the pointer to features needs to be stored in the buffer. Thus, the overhead of accessing the buffer is tiny.

Table 1
The speedup of different wait time (ticks), and different buffer sizes over SIFT based on 16 cores.

Ticks	Buffer size						
	8	12	16	32	64	128	256
5.0E+6	12.14	12.46	12.42	13.07	13.02	5.98	6.61
1.0E+7	13.18	13.37	13.26	13.53	13.38	12.61	8.62
5.0E+7	13.84	**14.07**	13.75	14.05	13.58	12.85	11.74

Table 2
The speedup of different wait time (ticks), and different buffer sizes over SURF based on 16 cores.

Ticks	Buffer size						
	8	12	16	32	64	128	256
5.0E+6	14.25	14.47	14.58	14.81	**15.14**	15.00	14.83
1.0E+7	14.64	14.90	14.72	14.88	14.95	15.05	14.79
5.0E+7	14.29	14.45	13.49	14.29	14.32	13.41	12.37

4.4. Throttling mechanism

The throttling mechanism must be taken into account to avoid too frequently thrashing threads between the two pipeline stages. We control the frequency of thrashing threads through two parameters: buffer size (size of shared buffer) and wait time (ticks from last adjustment of thread numbers), which are two important factors that may affect the overall performance. If the buffer size is too large, the adaptive pipeline will be insensitive to the changing of workloads, thus may not adjust system resource allocation in time. However, if the buffer is too small, AD-PIPE may be too sensitive. As a result, frequent changes of thread work will harm locality and cause extra overhead. Moreover, as another important parameter of our AD-PIPE, if the wait time is too short, the threads in different stages may be frequently adjusted, which will cause additional overhead. If the wait time is too long, some threads may wait too long time for the buffer which will also affect the performance. To decide the parameter of buffer size and wait time, we use the hybrid image set to evaluate the relation between performance and two factors on 16 cores. The data are shown in Tables 1 and 2. As the results shown, when the buffer size is 12 and the wait time is 5.0E+7 ticks, AD-PIPE of SIFT can achieve the best performance. And over SURF, the buffer size and wait time are 64 and 5.0E+6 ticks respectively. Therefore, we apply two configurations as the default parameters for AD-PIPE in the following parts of this paper.

Table 3
Two actions of thread transfer of AD-PIPE.

	Buffer state	Detection	Description
Action 1	Buffer (Full)	−1	+1
Action 2	Buffer (Empty)	+1	−1

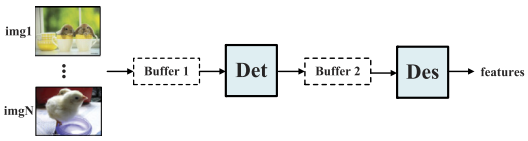


Fig. 10. The basic workflow of AE-PIPE.

5. Scheduling for variable workloads

Currently, power problems have become one of the most concerned issues in data center environments, which usually host hundreds or thousands of servers, such as *YouTube*, *Flickr*, and *Facebook* [6,26]. Considering a mass of image and video retrieval applications are running on data center servers day in and day out and their inputs vary in different time periods of a day, only accelerating the IFEAs, as the essential part of these programs, is far from enough to meet the power demand of modern data center. Therefore, it is appealing to design an efficient version for variable input workloads to achieve good power efficiency. In this section, we will first analyze the constraints of AD-PIPE in the environments with variable input workloads, and then give out the design of our energy-efficient adaptive pipeline parallelism (AE-PIPE).

5.1. Constraints for AD-PIPE

In AD-PIPE, we dynamically adjust threads between two pipeline stages to fully utilize computation resource. The actions for thread transfer are shown in Table 3. When the buffer between two stages is full for a while, the system performs Action 1 which transfer some threads in the detection state to description state. Similarly, Action 2 is performed to transfer threads from the description stage to the detection stage if the buffer is empty for a while.

Therefore, if input workloads are constant and the input buffer is always full, AD-PIPE can achieve balanced scheduling and work well. However, in practical conditions, input workloads of most server applications show very variable behavior. As an example, for a typical multimedia oriented website, the number of users drops gradually during the early morning (12–7 AM) and the afternoon (2–5 PM), while it climbs to a peak when users are in noon break (Noon-2 PM) or after work (6–9 PM) [33]. As a result, there are two constraints for AD-PIPE under real-world environments.

- AD-PIPE is always running for the worst cases since it assumes input workloads are constant and always full. All computation resources are set to make their best effort to do their jobs regardless of actual input workloads. Therefore, it is power inefficient in the most conditions.
- AD-PIPE will lead to unnecessarily thrashing threads in some conditions. For example, when input workloads decrease, the detection stage will have less work to do, and then the buffer will be empty. According to the scheme of AD-PIPE, threads will be transferred from the description stage to the detection stage. While the detection stage gets more threads, the buffer will be full. Some threads will be transferred to the description stage. Then the description stage gets more threads and has less work to do, so the buffer becomes empty again. Under this situation, the buffer state will change between empty and full frequently, which will result in thrashing threads unnecessarily.

As a result, many computation resources are wasted due to redundant running time caused by these constraints, which will lead to tremendous amounts of power waste. So we now analyze the wasted resources rate and identify our goal to extend AD-PIPE for power efficiency.

Suppose there is a set of cores C and for each core c_i, there are two kinds of running time of it: T_{W_i} and T_{I_i}. In addition, the whole execution time of the application can be divided in k non-overlapping intervals $N = \{n_0, n_1, \ldots, n_{k-1}\}$. In each interval n_i. There are two kinds of running time of it. One is effective working time T_E which is the actual working time for dealing with images and redundant time. The other is redundant time T_R, which is unnecessary, such as idle time or thrashing time. When T_R increases, the power efficiency will decrease because more computation resources will be wasted. The total wasted resource rate A which will lead to power inefficiency can be represented as follows:

$$A = \sum_{i=0}^{k-1} \sum_{j=0}^{|C|-1} \frac{T_{R_{i,j}}}{T_{R_{i,j}} + T_{E_{i,j}}}.$$

To achieve the goal of power efficiency, we should minimize the meaningless execution time to reduce wasted power consumption. In other words, we should minimize the total wasted resource rate A:

$$Minimize(A) \rightarrow 0.$$

Here, we also adopt the thought of the greedy algorithm as AD-PIPE. That is, in order to achieve the global minimized wasted resources rate A, our goal can be converted to achieve minimized wasted resources rate a_i in a certain interval n_i:

$$Minimize\left(a_i = \sum_{j=0}^{|C|-1} \frac{T_{R_j}}{T_{R_j} + T_{E_j}}\right) \rightarrow 0.$$

Therefore, some dynamic mechanisms should be involved to deal with variable workloads and achieve power-efficient.

5.2. Adaptive energy-efficient pipeline parallelism

Our goal is $A \rightarrow 0$, and that is to say improving power efficient as much as possible while maintaining the achieved performance. To achieve such a goal, we extend AD-PIPE to adjust the number of working threads according to the current input workloads, called Adaptive Energy-Efficient Pipeline Parallelism (AE-PIPE). We will first show the workflow of AE-PIPE and then discuss the fluidity of threads to achieve a better trade-off between energy and performance.

5.2.1. Workflow of AE-PIPE

For AD-PIPE, we schedule the threads through monitoring the buffer state between the detection stage and the description stage. However, such a design cannot work well when input workloads change. The hardware resource of a data center is generally set to be able to deal with the worst case. However, input workloads in most conditions are less than that of the worst case. Since fewer input workloads need fewer computation resources, redundant resources can be set in a low power state for power efficiency. Hence, we need to detect both of input workloads and inner workloads to achieve our goal in AE-PIPE.

There are several ways to monitor both kinds of workloads in different stages. We can detect the change of both workloads in every stage, which means each stage schedules threads according to the change of both workloads. However, such a design will result in chaos and the overhead will be larger for two reasons.

- The description stage is unable to judge which kind of workloads causes the buffer state changes. If current workloads of the description stage are decreasing, it can be raised from the imbalance between two stages or input workloads. When the buffer state changed, the description stage cannot figure out which condition happens since the description stage only cooperate with the detection stage.
- It would be difficult to synchronize between two stages if both of them schedule threads for input workloads. For example, when the workloads increase, the working threads should be increased. However, if both of them try to increase the number of threads, it will be difficult to decide how to allocate the available threads.

To avoid such constraints, we let the description stage only detect the inner workload change, and the detection stage is responsible for detecting both two workloads. In the design of AD-PIPE, we dynamically balance workloads between the detection stage and the description stage by transferring threads from the description state to the detection state and vice versa. When we aim to additionally maintain workloads of two stages in line with the actual input workloads, another mechanism should be extended to deal with the workload change. The basic workflow of AE-PIPE is shown in Fig. 10, and it is almost the same as AD-PIPE except that the input buffer (Buffer 1) is used to detect the change of input workloads. In other words, the description stage and the detection stage detect the imbalance between them as AD-PIPE. And the detection stage additionally detects input workloads and makes relevant adjustment.

5.2.2. Threads fluidity

In AD-PIPE, a thread is transferred only between the detection state and the description state. For AE-PIPE, in order to save energy and react to the input workload change, a new thread state, idle state, is introduced. When the thread state is idle, the CPU which executes this thread will shut down or be in sleep mode depending on the hardware implementation. In order to balance the number of threads in the idle state and working state according to input workloads, we get threads from idle state if we need more threads to handle current workloads and maintain the performance. Conversely, in order to save energy and decrease, we put threads into the idle state if there are redundant and wasteful threads.

After involving another thread state, the scheduling of threads becomes a little bit complex, the input buffer (Buffer 1) is involved to detect the change of input workloads and it has the same throttling mechanism as the origin one. Therefore, in the interval *i*, if the input buffer is full for a while, which means that input workloads raise and the current working threads cannot fully deal with it. Then, the threads in the idle thread state will be dragged to the detection state and start to work. Such a design can prevent AE-PIPE from losing performance caused by too many idle threads. After the threads go to work, they are controlled by AD-PIPE mechanism to achieve the outstanding acceleration on multi-core platform. When input workloads drop, the input buffer will be empty for a while, which means some threads are looping and consuming energy wastefully. Therefore, the redundant threads should take a nap and be transferred to the idle state at this time. The dynamic scheduling of threads is shown in Fig. 11, Buffer 1 is placed between the idle state and the detection state in the figure while it is actually between input images and the detection stage because detecting the input workload change is directly controlled through detecting the states of Buffer 1.

Combined with AD-PIPE, we summarize four thread transfer actions which are listed in Table 4. There are four actions for AE-PIPE scheduling. Action 1 and Action 2 which origin exists in AD-PIPE balance workloads between the detection stage and the description stage. Similarly, Action 3 and Action 4 are used to balance the number of idle threads and working threads. For the detection stage, it reacts to both two kinds of workloads and threads in it can be transferred to idle state or the description state correspondingly. For the description stage, it is only responsible for its own workloads. When workloads change, the description stage can only get or transfer its threads to the detection stage first. Then the detection stage decides whether some threads should be transferred to the idle state or wake up more threads from the idle state.

Table 4
Four actions of thread transfer show the changing number of different state threads.

	Buffer state	Idle	Detection	Description
Action 1	Buffer 2 (Full)	–	−1	+1
Action 2	Buffer 2 (Empty)	–	+1	−1
Action 3	Buffer 1 (Full)	−1	+1	–
Action 4	Buffer 1 (Empty)	+1	−1	–

Table 5
Three input image sets and the average feature point number in SIFT and SURF.

Image set	Size	SIFT features	SURF features
Small	640 ∗ 480	2722	999
Large	1600 ∗ 1200	4925	2132
Hybrid	small & large	3824	1566

With the help of threads fluidity, we decrease T_{R_i} while maintain T_{E_i} at the most suitable level for each interval *i*. Hence, AE-PIPE achieves a better trade-off between performance and energy.

6. Evaluation on multi-core platforms

In this section, we evaluate the performance of adaptive pipeline parallelism through comparing its results with those of other parallel schemes.

6.1. Experimental environment

Besides our adaptive pipeline parallel scheme, we also implement several other parallelism including image level (static and dynamic), scale level, block level, image-block parallelism (*the combination of image level and block level*) and static pipeline parallelism. The hardware platform is a 16-core server with 24 GB memory. Each processor is an Intel Xeon E7-4807 CPU with 1.87 GHz frequency. All the algorithms are compiled by GCC 4.4.0 with optimization argument -O2 under Fedora 11.

For SIFT, we use an open-source version provided by Rob Hess [16] as the baseline. For SURF, we also choose an open-source implementation, OpenSURF [11]. Various parallel versions of SIFT and SURF are implemented with *pthread* library [9]. We use three image sets for evaluation based on the feature point number, which affects the processing time of the IFEAs. The average point number of each set is listed in Table 5. The small image set is collected by K. Mikolajczyk [24] with modified 640 ∗ 480 image size. Images in the large image set are randomly downloaded from Google Image and each image has 1600 ∗ 1200 pixels. The hybrid image set is the data set including different image sizes. Each image sets includes 48 images. To reduce bias among different runs, each parallelized version is executed ten times, and the arithmetic mean is used. Besides different sets, we also checked our results through using some other images for our evaluation. The conclusions are similar with testing sets in the paper.

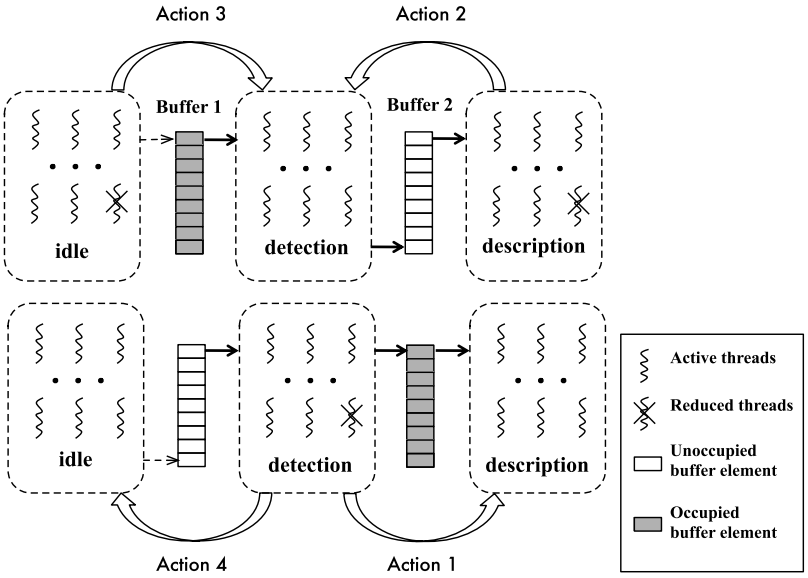


Fig. 11. Dynamic scheduling of AE-PIPE threads.

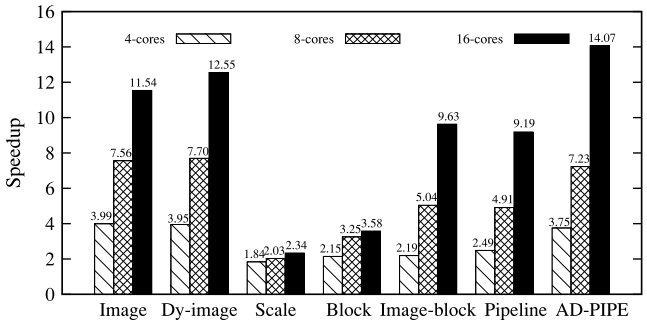


Fig. 12. The speedup of various SIFT parallelism for the hybrid image set.

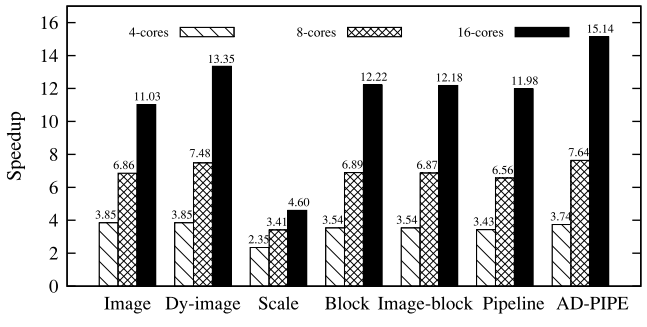


Fig. 13. The speedup of various SURF parallelism for the hybrid image set.

6.2. Performance of AD-PIPE

6.2.1. Performance of AD-PIPE

We first evaluate our AD-PIPE using the hybrid image set and compare it with the other parallelism. The speedup of processing time in all following experiments is measured by millisecond. The data are shown in Figs. 12 and 13. As data shown, our AD-PIPE has good scalability and it outperforms any other parallelism, with a speedup of 14.07X and 15.14X over SIFT and SURF on a 16-core machine. However, the speedup of AD-PIPE is a little lower than the ideal speedup. This is because the performance is limited by a series of factors such as the time of serial parts, the overhead of the adaptive adjustment and the communication synchronization.

As shown in Figs. 12 and 13, the performance of scale level parallelism is poor. They can only achieve a speedup of 2.34X over SIFT and 4.60X over SURF on 16 cores. The block level parallelism over SIFT is also poor, which has a speedup of 3.58X on 16 cores. The results are consistent with those of prior researches [37,14]. The major reason behind such results is that there are imbalanced workloads and frequent synchronization in such forms of parallelism. Block level parallelism of SURF requires less synchronization than that of SIFT, thus has a comparatively better performance of 12.22X speedup.

Image level parallelism has a good performance. When the number of cores is 4, we can see that the speedup is very close to the ideal speedup of 4X. On 16 cores, the speedup is 11.54X and 11.03X respectively over SIFT and SURF. The speedup of the dynamic version is 12.55 and 13.35 respectively. However, with the core number increasing to 8 and 16, the speedup is gradually away from the ideal speedup. This is mainly because of imbalance workloads on image level parallelism. The combination of image and block parallelism has good scalability, but the performance is limited by the deficiency of block level or image level parallelism. Although dynamic image level parallelism can overcome imbalance workloads to some extent, its results are also unsatisfied. The reason is that it is difficult for image level to know where and when the imbalance exists. Therefore, it is impossible for it to adjust its workloads among all threads based on the overall workload condition.

The static pipeline is designed to assign resources for two stages according to the workload of small images in SIFT (1:3 partition for detection and description stages).[1] We can see that the pipeline can only gain 9.19X and 11.98X speedup with the hybrid image set on 16 cores. Although they have acceptable scalability, the performance is not satisfied.

6.2.2. Evaluation on different input sets

To further illustrate the efficiency of our AD-PIPE, we further compare AD-PIPE with the parallelism with relatively good performance to show the speedup of the small image sets and large image sets. The results are evaluated by using 16 cores. As the data shown in Figs. 14 and 15, AD-SIFT also outperforms other parallelisms, which has a speedup of 13.45X for small image set

[1] We also test other partitions, such as 1:2, the results are similar. Since the space constraints, those results are not given out.

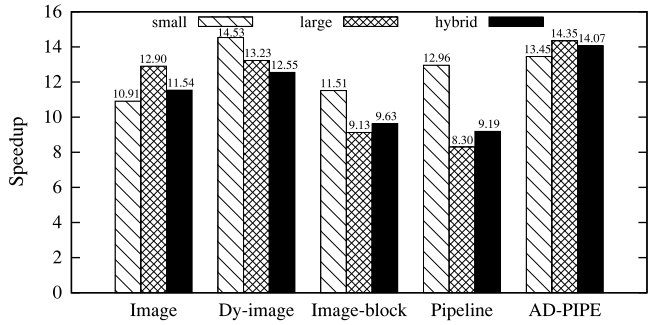


Fig. 14. The speedup of various SIFT parallelism for different image sets on 16 cores.

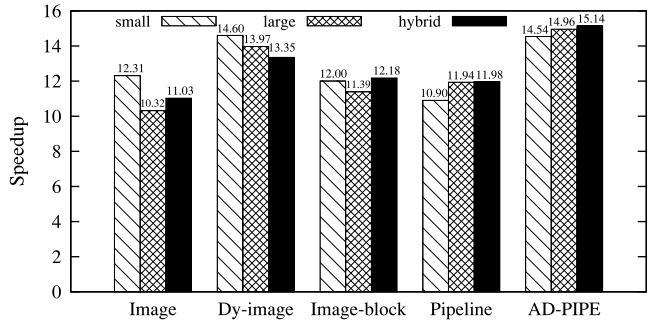


Fig. 15. The speedup of various SURF parallelism for different image sets on 16 cores.

and 14.35X for large image set. For small image set, only the results of dynamic image level parallelism is a little better than those of AD-PIPE. The reason is that the imbalance for small images is less serious than that of larger images. Therefore, dynamic image level parallelism works well. With image size increasing, on one hand, the imbalance condition become serious. It is difficult to adjust workloads for it. Moreover, with image size increasing, SIFT and SURF will suffer from the pressure of memory and bandwidth as analyzed in [14]. On the whole, AD-SIFT is the most efficient and scalable parallelism for all three image sets. When the input images are hybrid, the superior of AD-SIFT is more distinct, which makes it more applicable in the real-world environment. Similar situation is observed over SURF and AD-SURF also outperforms any other parallelism for all image sets.

6.3. Hyper-threading

Hyper-threading has become a popular design choice in multi-core design because of its efficiency and low hardware cost. Therefore, we further extend our implementation to 32 threads on 16 physical cores with hyper-threading. Figs. 16 and 17 show the hyper-threading performance of SIFT and SURF with 16 physical cores (*with hyper-threading, there are 32 logic cores*). When the configuration of hyper-threading is open, AD-PIPE also shows good scalability and beats any other parallelism for both SIFT and SURF. For SIFT, the speedup of AD-PIPE can further increase to 16.88X for small images, 16.38X for large images, and 14.92X for hybrid images. For SURF, the speedup of three image sets can also achieve a speedup of 20.33X, 20.36X and 20.47X respectively for different data sets, which exceed the ideal speedup and well prove the effectiveness of hyper-threading and our AD-PIPE implementation. The performance of image parallelism on the large and hybrid image set over SURF decreases compared to running on 16 physical cores without hyper-threading because of increased memory space and bandwidth requirements. This phenomenon does not appear in our AD-PIPE scheme, which also further confirms the effectiveness of our scheme.

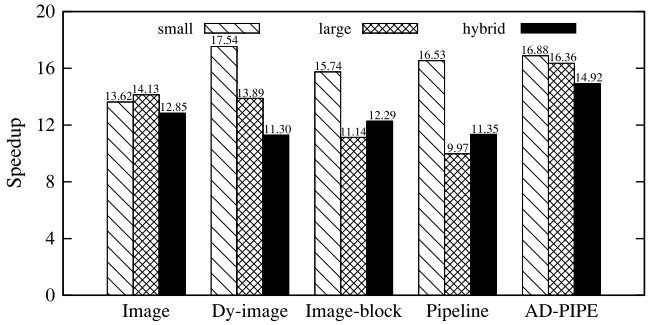


Fig. 16. The hyper-threading performance of SIFT parallelism on various image sets with 16 physical cores.

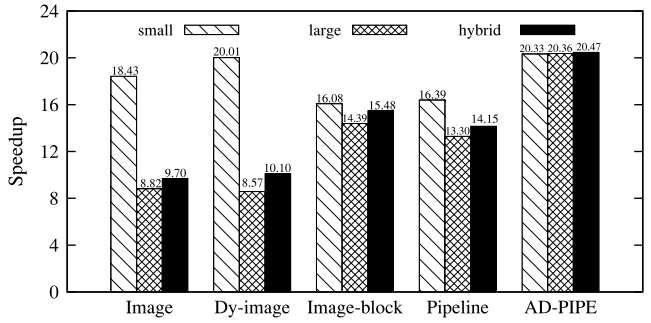


Fig. 17. The hyper-threading performance of SURF parallelism on various image sets.

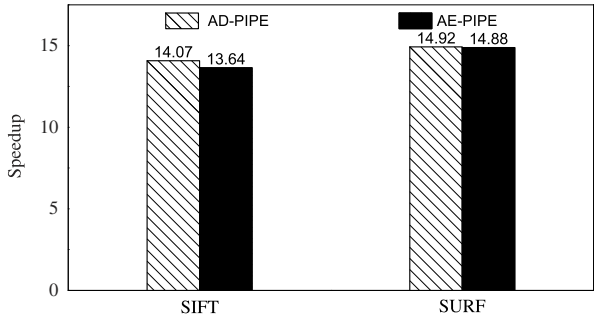


Fig. 18. The speedup of AD-PIPE and AE-PIPE for hybrid image set on 16 cores.

6.4. Performance of AE-PIPE

In this section, we first compare the performance of AE-PIPE with that of AD-PIPE. Then, we evaluate the power efficiency of AE-PIPE under different input workloads. We choose two configurations for the power consumption of idle time for CPU based on prior work [1,12]. One is 40% of busy time, which is a conservative configuration. The other is 10%, which is an aggressive configuration. The energy conservation percentage in the following experiments is measured through recording the idle time of all CPUs and calculating its related power consumption based on different configurations.

6.4.1. Overhead

We compare the speedup of AE-PIPE with AD-PIPE in the same environment shown in Section 6.2 under full input workloads. The input is the hybrid image set. As the data in Fig. 18 show, the performance of AE-PIPE is comparable with that of AD-PIPE. The slowdown of AE-PIPE is about 3% over SIFT and only 0.27% over SURF.

We also study buffer checking and scheduling overhead. We firstly evaluate the time used for each checking buffer and each

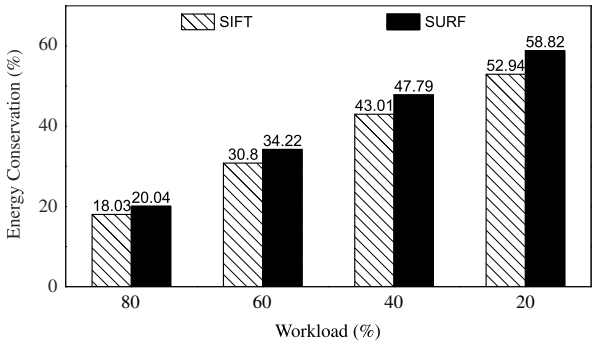


Fig. 19. The energy conservation performance of AE-PIPE for constant workloads when energy consumption of idle CPU is 10% of the normal state.

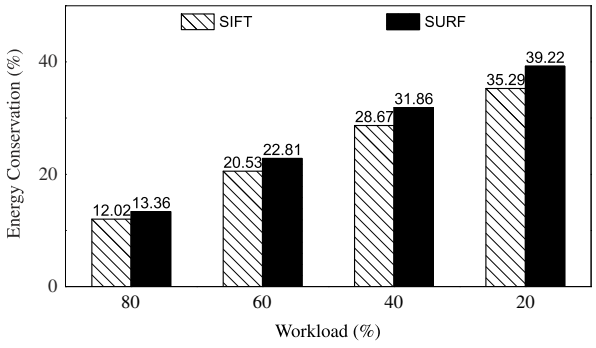


Fig. 20. The energy conservation performance of AE-PIPE for constant workloads when energy consumption of idle CPU is 40% of the normal state.

thread wake-up/sleep delay through constructing two test applications, which purely perform two operations respectively on a 16-core machine. We then run AE-PIPE for both AD-PIPE and AE-PIPE under mix workloads and count the frequency for buffer checking and thread status change. Based on such information, there is about 0.43% overhead for checking buffers and related scheduling logic. Therefore, the time overhead for monitoring and re-scheduling workloads is small and negligible.

Overall, the overhead of AD-PIPE is small and acceptable considering the energy conservation results of it as shown below.

6.4.2. Constant workloads

In order to show the basic capability of AE-PIPE for power efficiency, we first evaluate its power performance under constant workloads, which means the input workloads maintain a certain percentage of full workloads (maximum processing rate). We choose five constant configurations: 80%, 60%, 40% and 20% of full workloads, which reflects the effect of AE-PIPE under the stable user number or input workloads. The data are shown in Figs. 19 and 20. When input workloads are 80%, it saves 18.03% or 12.02% over SIFT and 20.04% or 13.36% over SURF. And when it comes to 20%, it saves 52.94% or 35.29% over SIFT and 58.82% or 39.22% over SURF. Based on such results, it is obvious that there is a definite inverse correlation between saved energy and input workloads, in line with expectations. Although constant workloads are ideal and seldom occur in the real case, it can be used to show the capability of AE-PIPE that it precisely saves the redundant energy on the beam.

6.4.3. Practical workloads

It is almost impossible to meet constant workloads in the real world. So we simulate four practical workloads as showed in Fig. 21. The width of a single block means the percentage of occupied time in the whole test and the number in the block means the

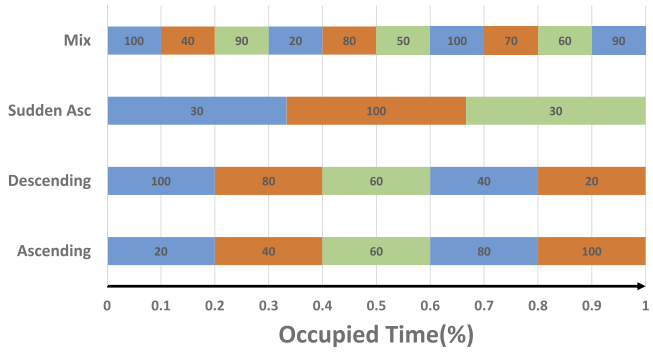


Fig. 21. Four practical workloads we are testing. The width of a single block means the percentage of occupied time in the whole test and the number in the block means the percentage of input workloads.

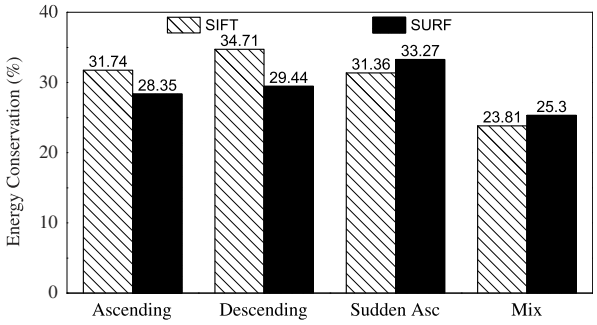


Fig. 22. The energy conservation performance of AE-PIPE for practical workloads when energy consumption of idle CPU is 10% of the normal state.

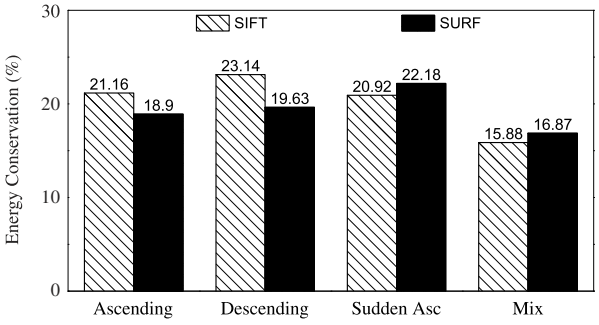


Fig. 23. The energy conservation performance of AE-PIPE for practical workloads when energy consumption of idle CPU is 40% of the normal state.

proportion of input workloads. The first two simulate the situation that the number of input images goes straight up or down such as workloads at early afternoon or after work in Section 5.1, and the next two shows there is a sudden peak period to simulate some occasional situation. Finally, the mix one means the workloads swing randomly to simulate unstable user flow application.

The results are shown in Figs. 22 and 23. For SIFT, it saves 31.74%, 34.71%, 31.36% and 23.81% under 10% idle power consumption and 21.16%, 23.14%, 20.92% and 15.86% under 40% idle power consumption for different workloads. For SURF, it saves 28.35%, 29.44%, 33.27% and 25.3% or 18.9%, 19.63%, 22.18% and 16.87% for different workloads under the best and worst assumption. Such results illustrate that AE-PIPE is efficient for power saving and the effect on SURF is slightly better than SIFT on the whole.

6.5. Accuracy

Since AD-PIPE and AE-PIPE only add buffers and related monitoring logic before detection stage or between detection and description stage, the execution of detection and description stage

itself is not affected. Under all kinds of workloads, AD-PIPE and AE-PIPE do not change the normal processing algorithm of SIFT and SURF so that the extracted feature points of AD-PIPE and AE-PIPE is the same as that of original sequential version of the IFEAs. Therefore, the accuracy results of AD-PIPE and AE-PIPE are the same with those of sequential versions.

7. Conclusion

In this paper, we first systematically analyze the characteristics, especially parallel constraints in the SIFT and SURF, two state-of-the-art local IFEAs. We find that imbalanced workloads and indeterminate time distributions are the major limitations for parallelizing them.

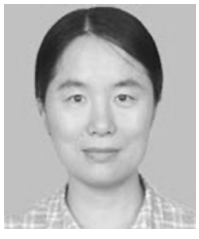
To alleviate the effect of the limitations, we designed and implemented an adaptive pipeline parallel scheme (AD-PIPE) for constant input workloads and a power efficient version (AE-PIPE) for variable input workloads. The algorithms can overcome the major parallel constraints in SIFT and SURF and achieve good performance and scalability.

Acknowledgments

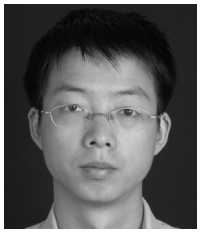
We are grateful to supports from the National High Technology Research and Development Program of China (No. 2012AA010905), the National Natural Science Foundation of China (No. 61370081), the Key Project of Major Program of Shanghai Committee of Science and Technology under Grant (No. 13DZ1108800). We would like to thank all our anonymous reviewers for valuable feedback on the paper.

References

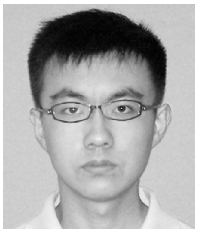
[1] L.A. Barroso, U. Holzle, The case for energy-proportional computing, Computer 40 (2007) 33–37.
[2] J. Bauer, N. Snderhauf, P. Protzel, Comparing several implementations of two recently published feature detectors, in: Proc. International Conference on Intelligent and Autonomous Systems.
[3] H. Bay, B. Fasel, L.V. Gool, Interactive museum guide: Fast and robust recognition of museum objects, in: Proceedings of the First International Workshop on Mobile Vision.
[4] H. Bay, T. Tuytelaars, L.V. Gool, Surf: Speeded up robust features, in: Proceedings of European Conference on Computer Vision, pp. 404–417.
[5] S.A. Berrani, L. Amsaleg, P. Gros, Robust content-based image searches for copyright protection, in: Proceedings of the 1st ACM International Workshop on Multimedia Databases, ACM, pp. 70–77.
[6] R. Bianchini, R. Rajamony, Power and energy management for server systems, IEEE Comput. 37 (2004) 68–74.
[7] M. Bicego, A. Lagorio, E. Grosso, M. Tistarelli, On the use of sift features for face authentication, in: Proceedings of Conference on Computer Vision and Pattern Recognition Workshop, p. 35.
[8] R.E. Bohn, J.E. Short, How much information, Technical Report, CISCO, 2009.
[9] D. Buttlar, J. Farrell, Pthreads Programming: A POSIX Standard for Better Multiprocessing, O'Reilly Media, Inc., 1996.
[10] P. Dreuw, P. Steingrube, H. Hanselmann, H. Ney, Surf-face: Face recognition under viewpoint consistency constraints, in: Proceedings of the British Machine Vision Conference.
[11] C. Evans, Notes on the opensurf library, Tech. Rep. CSTR-09-001, University of Bristol, 2009, January.
[12] X. Fan, L.A. Barroso, Power provisioning for a warehouse-sized computer, ACM SIGARCH Comput. Archit. News 35 (2007) 13–23.
[13] Z. Fang, D. Yang, W. Zhang, H. Chen, B. Zang, A comprehensive analysis and parallelization of an image retrieval algorithm, in: 2011 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, IEEE, pp. 154–164.
[14] H. Feng, E. Li, Y. Chen, Y. Zhang, Parallelization and characterization of sift on multi-core systems, in: Proceedings of IEEE International Symposium on Workload Characterization, pp. 14–23.
[15] I. Gordon, D.G. Lowe, What and where: 3D object recognition with accurate pose, in: Toward Category-Level Object Recognition, pp. 67–82.
[16] R. Hess, Sift library, 2010. http://blogs.oregonstate.edu/hess/code/sift/.
[17] Y. Ke, R. Sukthankar, Pca-sift: A more distinctive representation for local image descriptors, in: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004, Volume 2, IEEE, p. II–506.
[18] L. Ledwich, S. Williams, Reduced sift features for image retrieval and indoor localisation, in: Australian Conference on Robotics and Automation.
[19] L. Li, W. Xue, R. Ranjan, Z. Jin, A scalable helmholtz solver in grapes over large-scale multicore cluster, Concurr. Comp. Pract. E. 25 (2013) 1722–1737.
[20] D.G. Lowe, Object recognition from local scale-invariant features, Comput. Vis. 2 (1999) 1150–1157.
[21] D.G. Lowe, Distinctive image features from scale-invariant keypoints, Int. J. Comput. Vis. 60 (2004) 91–110.
[22] Y. Lu, X. Wang, W. Zhang, H. Chen, L. Peng, W. Zhao, Performance analysis of multimedia retrieval workloads running on multi- cores, IEEE Trans. Parall. Distr. Syst. (TPDS) (2016). http://dx.doi.org/10.1109/TPDS.2016.2533606.
[23] Y. Lu, X. Wang, W. Zhang, Y. Li, W. Zhao, Characterizing multi-media retrieval applications, in: The 44th International Conference on Parallel Processing (ICPP), pp. 270–279.
[24] K. Mikolajczyk, Local feature evaluation dataset, 2007. http://www.robots.ox.ac.uk/~vgg/research/affine/.
[25] K. Mikolajczyk, C. Schmid, A performance evaluation of local descriptors, IEEE Trans. Pattern Anal. Mach. Intell. 27 (2005) 1615–1630.
[26] P.N. Tseng, Y.L. Lin, W.H. Hsu, Interactive inquiry for object of interest in video playback by motion-augmented graph cut, in: Proceedings of the International Conference on Multimedia, pp. 811–814.
[27] J. Uijlings, A. Smeulders, R. Scha, Real-time bag of words, approximately, in: Proceeding of the ACM International Conference on Image and Video Retrieval, pp. 1–8.
[28] Y. Wan, Q. Yuan, S. Ji, L. He, Y. Wang, A survey of the image copy detection, in: 2008 IEEE Conference on Cybernetics and Intelligent Systems, IEEE, pp. 738–743.
[29] S. Warn, W. Emeneker, J. Cothren, A.W. Apon, Accelerating sift on parallel architectures, in: CLUSTER, pp. 1–4.
[30] Z. Weihua, W. Haojun, L. Yunping, C. Haibo, Z. Wenyun, A loosely-coupled full-system multicore simulation framework, IEEE Trans. Parall. Distr. Syst. (TPDS) (2016). http://dx.doi.org/10.1109/TPDS.2015.2455499.
[31] W. Xue, C. Yang, H. Fu, X. Wang, Y. Xu, L. Gan, Y. Lu, X. Zhu, Enabling and scaling a global shallow-water atmospheric model on tianhe-2, in: Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, IEEE, pp. 745–754.
[32] W. Xue, C. Yang, H. Fu, X. Wang, Y. Xu, J. Liao, L. Gan, Y. Lu, R. Ranjan, L. Wang, Ultra-scalable cpu-mic acceleration of mesoscale atmospheric modeling on tianhe-2, Comput. IEEE Trans. 64 (2015) 2382–2393.
[33] H. Yu, D. Zheng, B.Y. Zhao, W. Zheng, Understanding user behavior in large-scale video-on-demand systems, ACM SIGOPS Oper. Syst. Rev. 40 (2006) 333–344.
[34] K. Zhang, R. Chen, H. Chen, Numa-aware graph-structured analytics, in: ACM SIGPLAN Notices, vol. 50, ACM, pp. 183–193.
[35] N. Zhang, Computing parallel speeded-up robust features (p-surf) via posix threads, in: Emerging Intelligent Computing Technology and Applications, Springer, 2009, pp. 287–296.
[36] N. Zhang, Computing optimised parallel speeded-up robust features (p-surf) on multi-core processors, Int. J. Parallel Program. 38 (2010) 138–158.
[37] Q. Zhang, Y. Chen, Y. Zhang, Y. Xu, Sift implementation and optimization for multi-core systems, in: Proceedings of IEEE International Symposium on Parallel and Distributed Processing, pp. 1–8.



Yunping Lu is now a Ph.D. candidate in the School of Computer Science at Fudan University. Her research interests are in compilers, computer architecture, parallelization and systems software.

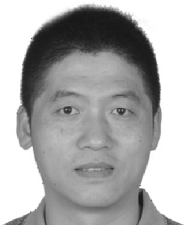


Yi Li received the Ph.D. degree in computer science from Fudan University in 2012. He is currently an assistant professor of Parallel Processing Institute, Fudan University. His research interests are in compilers, system software, computer architecture, and algorithm.



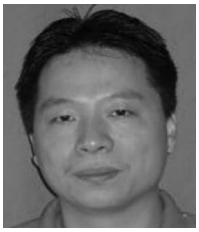
Bo Song is an undergraduate student in Software School, Fudan University. He is working in the Architecture Team of Parallel Processing Institute in Fudan University. His recent area is image retrieval algorithms.

His work is also related to computer architecture, simulation, execution variability and so on.



Weihua Zhang received the Ph.D. degree in Computer Science from Fudan University in 2007. He is currently an associate professor of Parallel Processing Institute, Fudan University. His research interests are in compilers, computer architecture, parallelization and systems software.

Haibo Chen received the B.Sc. and Ph.D. degrees in Computer Science from Fudan University in 2004 and 2009, respectively. He is currently a Professor in School of Software, Shanghai Jiao Tong University, doing research that improves the performance and dependability of computer systems. He is a senior member of the IEEE and the IEEE Computer Society.



Lu Peng received the Ph.D. degree in Computer Engineering from the University of Florida in Spring 2005. He is currently an associate professor in the Electrical and Computer Engineering department at Louisiana State University. His research focuses on computer architecture, memory hierarchy system, reliability, power efficiency and other issues in processor design.