# A improved GPGPU-accelerated parallelization for a rotation invariant Thinning algorithm

## Abstract

## Keywords

## 1. Introduction

第一段，细化定义介绍、细化算法的重要性，举例

Thinning is the process of reducing the thickness of each line of patterns to just a single pixel. Thinning is an important pre-processing step for many image analysis operation, such as image processing, character recognition, pattern recognition and so on.

第二段，细化算法的分类。细化算法分为两大类，parallel和sequential。Parallel下又能分为三类。

A comprehensive survey of thinning algorithms is described by Lam et al. in this paper, the author reviewed about 100 thinning algorithms. There two kinds of the thinning algorithm, the sequential thinning algorithm(STAs) and the parallel thinning algorithm(PTAs). As PTAs scan remove all the contour pixels that should be deleted in one iteration rather than only one pixel like STAs. So PTAs are usually faster than SPAS. There are three classes of PTAs: n-subiteration parallel Thinning algorithms, n-subfield parallel Thinning parallel algorithms and fully parallel Thinning algorithms.

第三段，随着图像大小的增大，目前细化算法的问题是运行时间过长，很难实现real-time。比如，Ahmed算法在细化2048\*2048大小图像需要十几秒。Kim在FPGA上实现了2D 细化算法并取得了很好的加速比，但是FPGA很难实现。Hu BingFeng在GPU上实现了3D细化算法，取得了152x的加速比。

With the increase of image size, such as satellite image and medi6cal image, result in the longer execution time of thinning algorithms. For example, thinning a 2D image(2048\*2048) takes about 17s on i7 CPU platform by using the A-W algorithm. It is a challenge to implement thinning algorithms in real-time. K.Kim et al implemented a PTA by using FPGA , but the implement is hard for general engineer and it limits the image size. Hu BingFeng et al implement 12- subiteration parallel 3D thinning algorithm on GPU and the speed-up achieved was 152x.

第四段，简单介绍cuda，并根据实验得出并行细化算法在GPU上能够得到很好的加速比。但是A-W算法的结果并不理想，因为该算法内存在很多分支，影响性能。

With the multi-thread parallel processing power, many problems, such as large data sets and intensive computation, can be resolved efficiently on GPGPU.as s modern GPU architecture, Compute Unified Device Architecture (CUDA) is supported by Nvidia GPUs for general-purpose parallel computing. The parallel thinning algorithms can be well implement on GPU computation plat form owing to its data parallelism. we implement several parallel thinning algorithms(PTAs) on GPU in figure 1.Figure 1 results show PTAs achieves a good performance on average 60 speed up improvement. But the speedup of A-W algorithm is not ideal. Because there are too many branch s in the algorithm flow which will lead a low warp efficiency.

在这篇论文中的主要贡献。1、证明PTAs在GPU上能够取得很好的加速比。2、在A-W的基础上，提出了一种新的并行策略，提高性能。3、我们讨论了性能与block大小、LookUp-Table的内存位置的关系。

In this paper, we manly focus on A-W algorithm, improved the algorithm with a look-up table, which decrease the branch. This paper makes the following contributions.

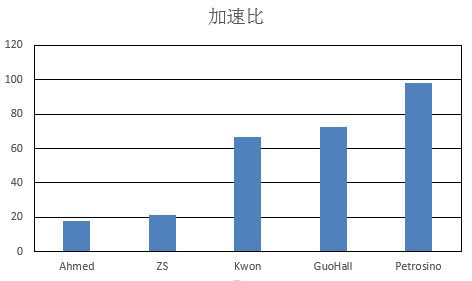
1. we implement several parallel thinning algorithms(PTAs) on GPU and proved the PATs could obtain a good speedup on GPU.

2. we present a novel and sample parallel strategy on A-W algorithm which could decrease the branch and logic complexity.

3. we discussed in detail about the relationship between the speedup

论文余下部分的组织。

The rest of the paper is organized as follows. Section 2 briefly describe the A-W algorithm. Then section 3 illustrates the detailed design of A-W algorithm on CUDA. Experimental analysis and results are shown in Section4 and Finally we offer the future work and conclusion in Section5.



## 2. A rotation invariant ruled-based Thinning Algorithm

A-W algorithm is the first thinning algorithm focus on rotation invariant. The A-W algorithm is a iteration algorithm. At each iteration, the target pixel is divided into four classes through its 3\*3 neighborhood. There are three steps in A-W algorithm.

For every pixel ***p*** do the following:

**Step1**. If p belongs to \*\*\*, stop calculations for this pixel. Otherwise, go to step2.

**Step2**. If p belongs to \*\*\*, delete w. Otherwise, go to step3.

**Step3**. Apply the 20 templates to ***p***.

## 3. A-W Thinning Algorithm On CUDA

We divide the A-W algorithm into several parts, and design a novel structure on CUDA. The whole algorithm includes host-side(CPUs) and device –side(GPU). It can be seen in fig.2 that the host side mainly calculate the Lookup-Table and read image, while the main part are parallelized on device-side, including step 1, step 2, step 3. In this paper, we focus on step 3 and apply lookup-table on step 3.

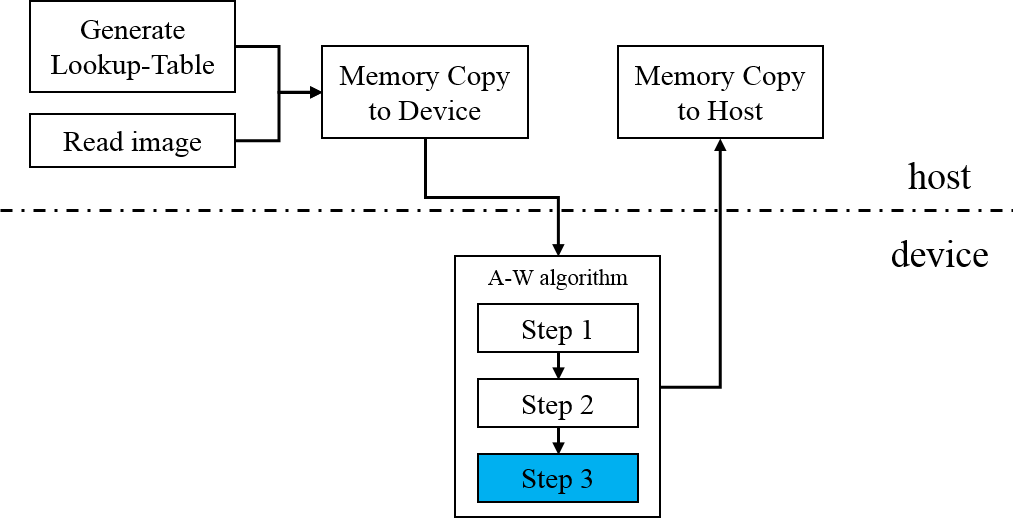


Figure 2. structure of A-W thinning algorithm on cuda

### 3.1 Definition

For every pixel ***p***,

八邻域Np

**Definition 1** N(p) is the set of all neighbors of p. Neighbors having even numbers(p[0], p[1], p[2], p[3], p[4], p[5], p[6], p[7], p[8]).

|  |  |  |
| --- | --- | --- |
| P[0] | P[1] | P[2] |
| P[3] | P | P[4] |
| P[5] | P[6] | P[7] |

Figure 4. Neighbors of p

标点的重量值。WN(p)

**Definition 2. Weight Number** of p, WN(p), is defined:



For neighbors of p, the weight numbers are among 0 and 255.

**Definition 3.** **Template** is the fixed set of neighbors of target pixel. For example, in A-W algorithm, If the neighbors of p match the template in figure 5, then delete p.

|  |  |  |
| --- | --- | --- |
| 1 | X | 0 |
| 1 | 1 | 0 |
| 1 | 1 | x |

Figure 5. One template in A-W algorithm.

### 3.2 template-to-lookup table

在GPU内，分支对性能的影响。

In GPU, threads are grouped into warps. Warp Instruction is the execution unit of GPU, which means all the 32 threads execute the same instruction at any clock. Branch divergence has a significant impact on the performance of GPU programs. Branch divergence can hurt performance due to lower utilization of the execution units, which cannot be compensated for through increased levels of parallelism[1]. For example, in figure 3, where half of the warp threads evaluate the branch condition to true, the utilization of the execution units is only 50%.

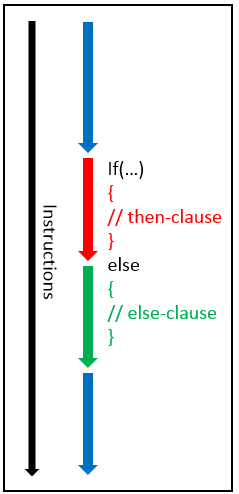


Figure 2 example of branch

在A-W算法中，step 3理论上产生分支的最大深度为20，这大大降低了性能。

in A-W algorithm, there are 20 rules in step 3. It means, when executing step 3, in the worst case, the utilization of the execution units is only 1/20 = 5%.

在本章节中，在A-W算法的基础上，提出了一种novel的方法，能够将template 转化为LookUp-Table

In this section, we propose a novel method based on A-W algorithm. This method could transform templates into lookup-table. The code in Fig show part of the code for generate lookup table.

char Lookup-Table[2^8] = {0};

for(ti in templates) {

if(match(Np, ti)) {

Lookup-Table[WN(p)] = 1;

}

}

那么A-W算法的step 3就可以转变为，如下。

Then we could transform 20 templates in to a 256 size lookup-table and we can design the step 3 as follows.

|  |  |
| --- | --- |
| 原先代码  if (match(Np, t[0]) || match(Np, t[1]) ||  match(Np, t[2]) || match(Np, t[3]) ||  match(Np, t[3]) || match(Np, t[5]) ||  match(Np, t[4]) || match(Np, t[7]) ||  match(Np, t[8]) || match(Np, t[9]) ||  match(Np, t[10]) || match(Np, t[11]) ||  match(Np, t[12]) || match(Np, t[13]) ||  match(Np, t[14]) || match(Np, t[15]) ||  match(Np, t[16]) || match(Np, t[17]) ||  match(Np, t[18]) || match(Np, t[19]) ||  ) {  delete p;  } | LUT代码  int WN = isHigh1(x1) \* 1 + isHigh1(x2) \* 2 +  isHigh1(x3) \* 4 + isHigh1(x4) \* 8 +  isHigh1(x5) \* 16 + isHigh1(x6) \* 32 +  isHigh1(x7) \* 64 + isHigh1(x8) \* 128;    if (LUT[WN] == 1) {  delete p;  } |

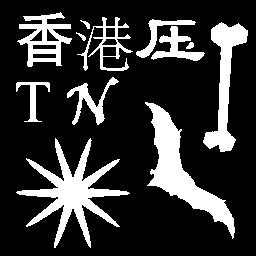
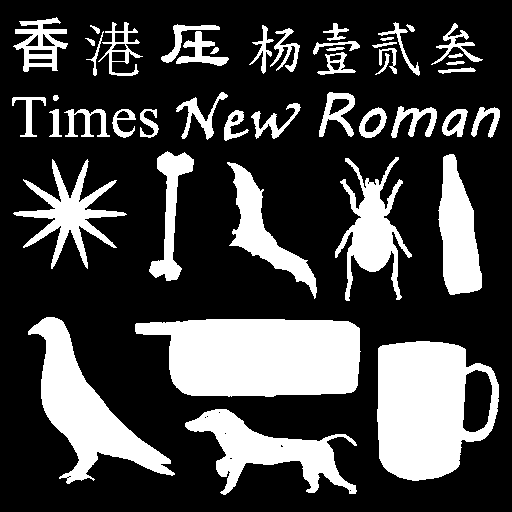
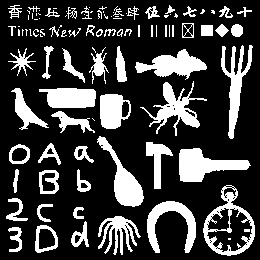
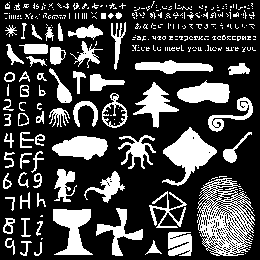
## 4. Experimental Results and Analysis

介绍实验图像和实验平台。

To evaluate our approach, we have chosen an Intel Core(TM) i7-2600 3.40GHz CPU and a NVIDIA GTX TITAN graphics card, using CUDA version 7.0. Four images are used in our experiment. Fig show the images and table show the detail of the images.

|  |  |  |
| --- | --- | --- |
| 图像ID | 图像大小 | 图像内点数 |
| IA | 256\*256 | 11105 |
| IB | 512\*512 | 63150 |
| IC | 1024\*1024 | 215767 |
| ID | 2048\*2048 | 944177 |

Table the detail of images

实验一，正常GPU加速和look up table方法加速的加速比。随着图像大小的增大，加速比越来越明显。

Firstly, we compare the performance of the origin A-W algorithm and TTL. As shown in fig , the average speedup of TTL achieve 30x(maximum 40.5x improvement) and better than origin on average 176%. With increasing image size, speedup get higher.

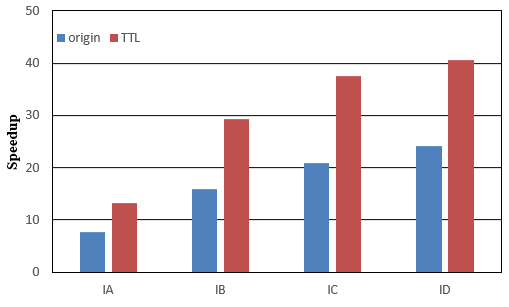


Fig speedup comparison of four images between origin and TTL.

实验二：应用TTL策略与直接GPU加速的加速比，并对比nvvp统计的数据，进行分析

We get the detail statistical data when executing the kernel using the NVIDIA Visual Profiler(NVVP). NVVP is a cross-platform performance profiling tool that delivers developers vital feedback for optimizing CUDA C/C++ applications. We selected four criteria, Achieved Occupancy, Warp Execution Efficiency, Control-Flow Instructions, IPC. As shown in table, the TTL could decrease Control-Flow Instructions 465% which lead the higher Achieved Occupancy, higher Warp Execution Efficiency and higher IPC.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Achieved Occupancy | Warp Execution Efficiency | Control-Flow Instructions | IPC |
| Origin | 0.534 | 37.3% | 3921276 | 1.859 |
| TTL | 0.607 | 59% | 842756 | 2.164 |

Table the detail statistical data from NVVP

实验三：分析加速比和block大小的关系

Then, we analysis the relationship between performance and Block size. Threads within a block can consume a fixed size registers and shared memory. The block size will impact the occupancy and the unsuitable block size will lead low utilization. The result is showed in fig. it shows that 32\*4 is the best block size for A-W algorithm on GeForce GTX TITAN.

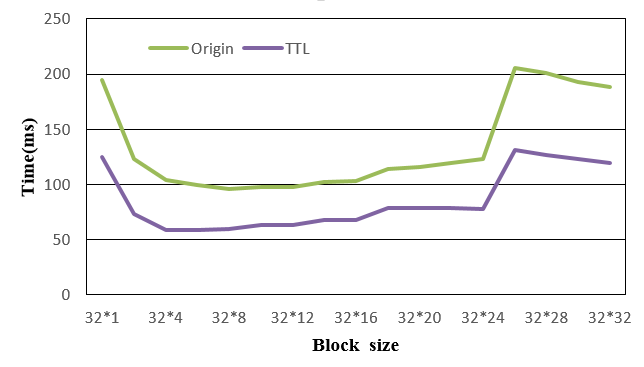


Fig relationship between block size and speedup

实验四：对比LookUp Table在Constant、Global、Shared内的速度

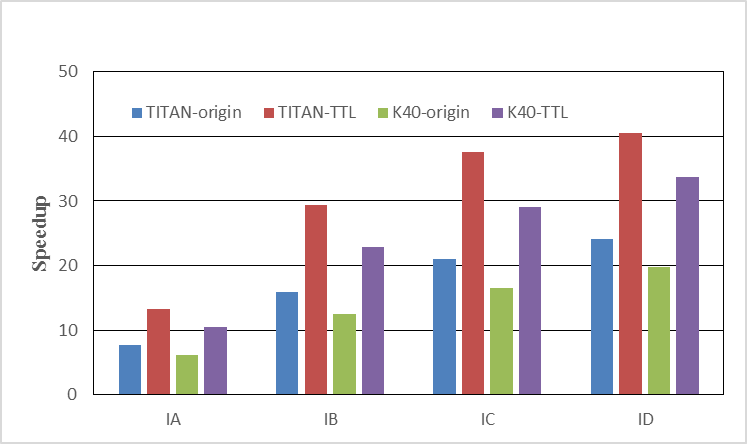
In GPU, global memory is off-chip. Shared memory and constant memory are on-chip and faster than global memory. We analysis the speedup when LUT located in global, shared and constant. As table shown, the speedup in global is almost equal the speedup in constant and the best block size of both is 32\*4. That is because the throughput of constant memory is 4B per clock per SM. So, unless the entire warp reads the same address, replays are need. The speedup in shared memory is slower and the best block is 32\*16. In TTL, each thread only read once. Though the access time of LUT is faster, more time to copy the LUT from global to shared.

|  |  |  |
| --- | --- | --- |
|  | Best Performance | Block size |
| Global | 58.1382 | 32\*4 |
| Shared | 69.291 | 32\*16 |
| Constant | 58.48 | 32\*4 |

Table the speedup when LUT located in global, shared and constant.

实验五：对比K40和TITAN上的加速比

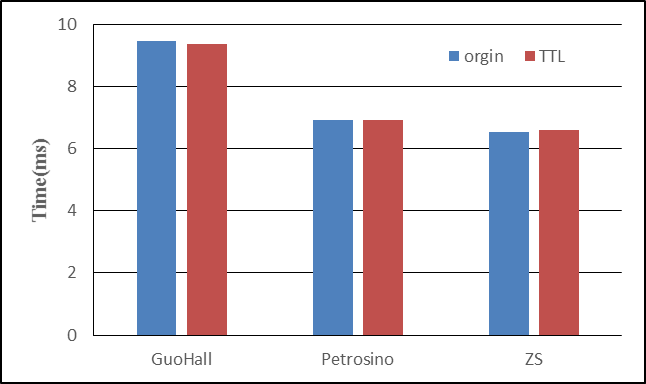
In addition, we executed the kernels on diffident GPU, Tesla K40c and GeForce GTX TITAN. Figure shows the speedups.



Fig

实验六：说明TTL具有一定的开销，并不是在所有的细化算法上都有很好的效果。该方法有一定的消耗，所以并不是对所有的并行细化算法都有很好的效果。

Finally, we will discuss the consumption of TTL. TTL can improve performance of A-W algorithm on CUDA by decreasing the branch in step 3, but there are Time consuming in copy the LUT from host to device and calcite the WN(p) in each thread. We apply TTL to several others thinning algorithms with less branch. As table shown, TTL get negligible performance improvement.



Fig

分析占用时间的部分，如果拷贝占的时间长，那么可以说随着GPGPU架构的发展，数据访问速度会越来越快，该方法的前景也**会**越来越好。

## 4 future work

该方法不仅适用于细化算法，在图像处理算法领域，只要满足根据领域特征进行复杂判断的图像算法都可以用该方法进行GPU加速

We call A-W thinning algorithm “branchs-algorithms” which always need the feature of p’s neighbors to make decision. In these algorithms, TTL will get a good speedup on CUDA.

## 5. Conclusion

In this paper, firstly, we proved that thinning algorithms are suitable to implement on GPU. Second, we propose a novel parallel strategy based on A-W thinning algorithm using CUDA. This strategy can decrease the branch in each thread by transform Templates into Lookup Table. Then we analysis the relationship between speedup with block size. For data allocation, we compare the speedup of data storage in global, constant and shared memory. The experiment result shows the speedup of the proposed strategy can reach 40.5x . Moreover, we analyze the consumption of TTL and we conclude that our approach is not only more suitable with A-W algorithm on larger image, but also other “branchs-algorithms”.

## 7. Reference

[1] Han T D, Abdelrahman T S. Reducing branch divergence in GPU programs[C]//Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units. ACM, 2011: 3.

[2] Ahmed, M. & Ward, R. A rotation invariant rule-based thinning algorithm for character recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 24, 1672-1678 (2002).

[3] Kim, K.H.; Pham Cong Thien; Seung Hun Jin; Kim, D.K.; Jae Wook Jeon "Dedicated parallel thinning architecture based on FPGA", Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on, On page(s): 208 – 213

[4] Guo, Z., & Hall, R. W. (1989). Parallel thinning with two-subiteration algorithms. Communications of the Acm, 32(32), 359-373.

[5] Petrosino, A. ,., & Salvi, G. ,. (2000). A two-subcycle thinning algorithm and its parallel implementation on simd machines.. IEEE Transactions on Image Processing A Publication of the IEEE Signal Processing Society, 9(2), 277 - 283.

[6] Wang, P. S. P., & Zhang, Y. Y. (1996). A Parallel Thinning Algorithm with Two-Subiteration that Generates One-Pixel-Wide Skeletons. Pattern Recognition, International Conference on(Vol.4, pp.457). IEEE Computer Society.

[7] Zhang, T. Y., & Suen, C. Y. (1984). A fast parallel algorithm for thinning digital patterns. Communications of the Acm, 27(3), 236-239.

[8] Lam, L., Lee, S. W., & Suen, C. Y. (1992). Thinning methodologies-a comprehensive survey. IEEE Transactions on Pattern Analysis & Machine Intelligence*,* *14*(9), 869-885.

[9] Hu, B., & Yang, X. (2013). GPU-Accelerated Parallel 3D Image Thinning. High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC\_EUC), 2013 IEEE 10th International Conference on(pp.149 - 152). IEEE.