# Additive Hough Transform on Embedded Computing Platforms

Shantanu S. Sinha
Indian Institute of Technology Bombay
Mumbai, India
shantanu.s.sinha@iitb.ac.in

R. K. Satzoda, S. Suchitra, T. Srikanthan
Nanyang Technological University
Singapore
{satz0002, such0004}@e.ntu.edu.sg, astsrikan@ntu.edu.sg

***Abstract*— The Hough Transform (HT) is one of the most widely used feature extraction techniques in real-time applications, like lane departure warning, surveillance etc. Most existing HT implementations are serial in nature, resulting in high computation time. Recently, we proposed the Additive Hough Transform (AHT), which achieves angle and block level parallelism in HT computation. In this paper, we evaluate the performance gains offered by AHT in both serial and parallel configurations by implementing it on different computing platforms such as multicore processors, GPUs and FPGAs. It is shown that AHT offers significant performance gains over existing implementations of the HT in both serial and parallel configurations on a wide range of embedded platforms.**

## I. INTRODUCTION

The Hough Transform (HT) is a widely used feature extraction technique to detect instances of parametric curves using a voting procedure in parameter space [2]. The HT is computationally expensive as it involves computation of transcendental functions and this has limited its use in real-time capable systems. Angle-level parallelism has been explored for HT computation in [7], [8] to improve its performance, wherein HT for different angle spaces are computed in parallel. However, such architectures still require the edge map to be read in a raster scan fashion and, hence, most existing parallel implementations are constrained by serial reading of the image. Most GPU implementations of HT involve pixel-level parallelism [9], [10]. While pixel-level parallelism does prove to be effective on high-end GPUs, it is not feasible on commercially available commodity GPUs with less streaming multiprocessors, as it requires concurrent execution of a large, suboptimal number of threads, leading to poor performance.

In [4]-[6] and [8], attempts have been made to reduce the computational complexity involved with the evaluation of trigonometric quantities by replacing them with simple shift and add operations using Coordinate Rotation Digital Computers (CORDIC) [3]. However, CORDIC, being iterative in nature can pose to be a bottleneck in the overall performance of the HT. Also, most of the CORDIC-based HT implementations are serial in nature, thereby limiting the achievable speedup.

We proposed a novel parallel implementation of the Hough Transform called the Additive Hough Transform (AHT) in [1], that exploits additive properties of the HT to replace trigonometric operations with simple additions in a block based fashion resulting in angle and block level parallelism. It was shown in [1] that the AHT is capable of achieving a speedup by orders of magnitude for any image size, and is suitable for porting onto massively parallel systems like FPGAs. In this paper, we demonstrate the feasibility of implementing the AHT on multiple embedded platforms. Further, it is shown that AHT outperforms its existing counterparts in both serial and parallel configurations with respect to execution time and resource utilization. The rest of the paper is organized as follows. In Sec. II, the AHT is described briefly. The different implementations of AHT and existing conventional HT that were developed for evaluation, are described in Sec. III. A detailed performance evaluation of the implementations is presented in Sec. IV followed by conclusions in Sec. V.

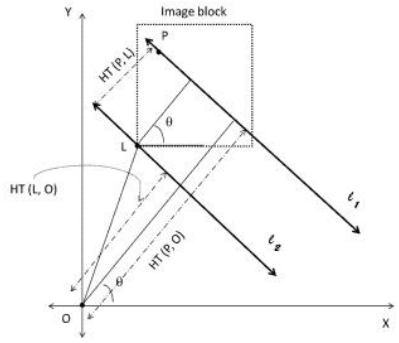


Figure 1. Additive property of the Hough transform. L is the local origin of the block that contains P, and O is the global origin of the image. $\ell_1$ and $\ell_2$ are parallel lines passing through P and L respectively.

## II. ADDITIVE HOUGH TRANSFORM

In this section we briefly discuss the block-based parallel Additive Hough Transform (AHT) technique. The HT maps a pixel $P(x,y)$ in the Cartesian space into a sinusoidal curve in the Hough space indexed by $\rho$-$\theta$ [2] using

$$\rho = x\cos(\theta) + y\sin(\theta) \quad (1)$$

where $\rho$ is the length of the perpendicular to a line passing through $P$ and subtending an angle $\theta$ with the x-axis.

In [1], we proposed to divide the image edge map into rectangular blocks of binary pixels. It was shown that the HT of any edge pixel in the block can be calculated as the sum of the HT value of the edge pixel with respect to (w.r.t) the local origin of the block it belongs to and the HT value of this local

origin w.r.t the global origin of the image. This property is called the additive property of HT represented as

$$HT(P, O) = HT(P, L) + HT(L, O) \quad (2)$$

where $L$ is the local origin of the image block which contains the pixel (Fig. 1). Henceforth, *HT(P, L)* will be referred to as the Local Hough Transform (LHT) of the pixel P and *HT(L, O)* as the Global Hough Transform (GHT) of the local origin $L$. In [1], the additive property of the HT was exploited to develop angle and block level parallel architectures, wherein multiple edge blocks were processed in parallel. More details can be found in [1].

## III. Proposed AHT Implementations

We implemented variants of AHT on three different computing platforms – programmable CPUs, graphics processing units (GPUs) and reconfigurable hardware (FPGAs). On each of these platforms, we implemented both serial and parallel versions of AHT in addition to serial and parallel versions of the regular HT to quantify the performance benefits offered by AHT with respect to the regular HT. We list the different implementations in this section.

### A. Processor-based Implementations

Multicore CPUs and DSPs like Infineon TriCore [11] are commonly used by in applications like Advanced Driver Assistance Systems (ADAS), which are heavily dependent on HT-based object detection. Achieving real-time performance is the primary bottleneck in such computing systems. We propose two variations of AHT and compare them with conventional non block-based implementations. We implemented the following three versions of the conventional HT:

- **i.** The most conventional serial HT implementation approach [2].
- **ii.** The OpenCV implementation of HT: This is a conventional HT implementation as in (i), but caches sine and cosine values at runtime.
- **iv.** A modified parallel version of (i) which reads sine and cosine values from LUTs. Multiple rows of the edge map are processed in parallel.

The following two versions of AHT were implemented on processor-based platforms:

- **iii.** A block-based serial version of AHT, which reads LHT values directly from LUTs. GHT values are computed using (1), reading sine and cosine values from LUTs.
- **v.** A parallel version of AHT with block-level parallelism. Both LHT and GHT values are read directly from LUTs.

### B. GPU-based Implementation

GPUs are widely used in graphics and vision based applications owing to their highly parallel architecture. We implemented AHT in the CUDA programming language to evaluate its performance on GPU architecture. The CUDA thread hierarchy divides threads into blocks and then organizes the blocks into a grid. Each block has its own shared memory that is accessible by all threads within that block.

In implementation **(vi)**, we implement AHT such that each block of threads of the CUDA hierarchy operates on a single image block. Additionally, given the highly parallel nature of the GPU architecture, a controlled amount of angle-level parallelism in HT computation is also introduced to further maximize computational efficiency. GHT values are read directly from LUTs in shared memory within the blocks, while LHT values are read from LUTs in global memory.

### C. FPGA-based Implementations

We evaluated the performance of AHT on FPGA platforms against existing hardware architectures for HT. We implemented the following HT architectures that exploit the angle parallelism in HT, i.e. reading the edge map serially in a raster scan fashion but computing HT for multiple angles in parallel:

- **vii.** A CORDIC engine [3] in vector rotation mode is used for HT computation as described in [5].
- **viii.** Sine and cosine values of the angles HT is being computed for are read from LUTs instead of using a CORDIC engine.

In our implementations of AHT we divide the $n \times n$ image

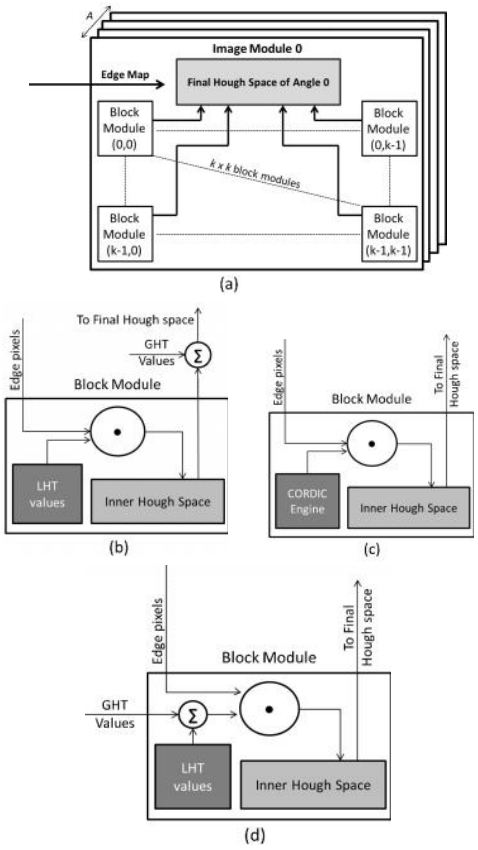


Figure 2. (a) High-level overview of AHT implementations in hardware. (b) Block modules in implementation (ix). (c) Block modules in implementation (xi). (d) Block modules in implementation (x)

into $k \times k$ blocks of $m \times m$ pixels each. Each block is processed in parallel. Our architecture (Fig. 2(a)) consists of one higher-level module per angle, accepting the complete

edge map as input and filling the final Hough space. We shall refer to these modules as image modules and for computation of the HT for $A$ angles, we would need $A$ image modules. Additionally, each image module comprises $k \times k$ lower level modules, each of which take one block of the edge map as input and store temporary inner Hough spaces. We shall refer to these as block modules. The inner Hough spaces contain the HT of the individual blocks and must be summed to compute the final Hough space. In Fig. 2(b, c, d), ● checks for edge pixels in the input data stream and ∑ represents an adder. The following versions of AHT were implemented:

**ix.** Block modules compute only the local HT of the block. The inner Hough spaces hence store only LHT values and, hence, must be summed serially with the corresponding GHT values before being transferred to the final Hough space. (Fig. 2(b))

**x.** Block modules compute the final HT of all pixels in the block. The inner Hough spaces would then be larger than in (ix), but can be directly transferred to the final Hough space without any need for further addition (Fig. 2(d)).

**xi.** This version is the same as (x), except for the use of CORDIC engines to compute the LHT instead of LUTs (Fig. 2(c)).

## IV. Performance Evaluation

In this section, the performance evaluation of the different implementations is discussed in detail. Implementations (i) to (v) were implemented on a 1.6 GHz quad-core Intel Core i7 Q720 CPU. The CPU allows execution of eight threads in parallel which was exploited by the parallel implementations (iii) and (v) using the OpenMP multi-processing API [6]. Algorithm (vi) was implemented in the CUDA programming language [5] on an NVIDIA GT630M GPU with 96 processing cores and 2GB of device memory. VHDL implementations of (vii) through (xi) were synthesized on a Xilinx Virtex6 XC6VLX760 FPGA.

Table I lists the different implementations and their corresponding timing complexity. LUT sizes and processing times are given for $n \times n$ images which, in AHT implementations, are divided into $k \times k$ blocks, each comprising $m \times m$ pixels. It is assumed that the HT is computed for $A$ angles and that there is no hardware constraint on the maximum number of threads allowed in any implementation. The block-level parallelism offered by AHT reduces the time complexity from $O(n^2)$ to $O(m^2)$. Since we generally choose $m$ to be at least an order of magnitude smaller than $n$, it can be seen that AHT implementations perform orders of magnitude faster than the regular HT.

We tested all the implementations from Table 1 on $256 \times 256$ edge maps ($n = 256$) and, for AHT implementations, we chose $k$ to be 32, which gives $m = 8$. Considering that the computation time of software implementations can vary with edge content in input image, the performance of the software versions, i.e. (i) to (vi) was evaluated by running the implementations for a set of benchmark images. Fig. 3(a) compares the parallel software implementations of AHT and the regular HT for 35 benchmark images. Both versions of AHT perform an average of 160% faster than the row-wise parallel HT implementation (iv) for all images. Fig. 3(b) compares the serial software implementations. AHT (iii) is found to be the least computationally demanding among the serial implementations and, on average, was found to be 140% faster than the OpenCV implementation of HT even on serial architecture. It is noteworthy that although AHT is designed for parallel hardware platforms [1], yet it offers significant performance gains on both serial and parallel software implementations for single-core CPUs and DSPs. This is primarily because of the low computation cost of the operations involved in AHT as compared to conventional HT.

The GPU and CPU versions of AHT take comparable amounts of time, even on a low-end commodity GPU like the GT630M. The block-based nature of AHT requires concurrent execution of fewer threads than complete pixel-level parallelism, making it feasible on lower-end graphics hardware as well. We achieved speeds in excess of 240 frames per second on the GT630M. On faster scientific computing GPUs with more streaming multiprocessors we can expect much faster execution due to the scalability of AHT.

Fig. 4 compares the different hardware implementations of AHT and the regular HT. AHT implementations (ix) and (x) are able to achieve the highest clock speeds, in excess of 350MHz. Due to the block-level parallelism offered by AHT in addition to the angle parallelism, implementations (ix), (x) and (xi) require only about 10% of the cycles required by (vii) and (viii). This reflects in the computation time per image if we set the clock speed to a fixed value of 200 MHz for all implementations.

TABLE II. Implementations Tested

| Description | Level of Parallelism | Running Time | Size of LUTs |
|---|---|---|---|
| ***Single Core CPUs/ DSPs*** | | | |
| xii. Conventional HT | None | $O(n^2A)$ | 0 |
| xiii. OpenCV HT | None | $O(n^2A)$ | 2A |
| xiv. AHT (serial) | None | $O(n^2A)$ | $(m^2 + 2)A$ |
| ***Multicore CPUs/DSPs*** | | | |
| xv. HT with naïve parallelism | Row | $O(n^2A)$ | 2A |
| xvi. AHT (parallel) | Block | $O(m^2A)$ | $(m^2 + k^2)A$ |
| ***Standard Parallelized Systems*** | | | |
| xvii. AHT (CUDA) | Block, angle | $O(m^2)$ | $(m^2 + k^2)A$ |
| ***Reconfigurable Platforms*** | | | |
| xviii. HT with CORDIC | Angle | $O(n^2)$ | 0 |
| xix. HT with LUTs | Angle | $O(n^2)$ | 2A |
| xx. AHT with small inner Hough spaces | Block, angle | $O(m^2)$ | $(m^2 + k^2)A$ |
| xxi. AHT with large inner Hough spaces | Block, angle | $O(m^2)$ | $(m^2 + k^2)A$ |
| xxii. AHT with CORDIC | Block, angle | $O(m^2)$ | 0 |

In hardware applications, if the range of angles for which the HT is to be computed is known a priori, LUTs can be used and, in such cases, AHT implementation (x) was found to be 35× faster than the comparable conventional HT implementation (viii) at the highest allowed clock speed. In

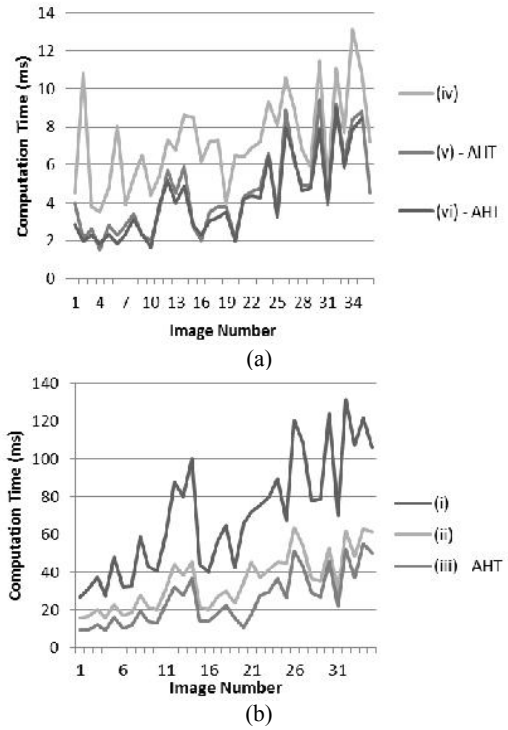


Figure 3. Comparison of (a) parallel implementations and (b) serial implementations in software.

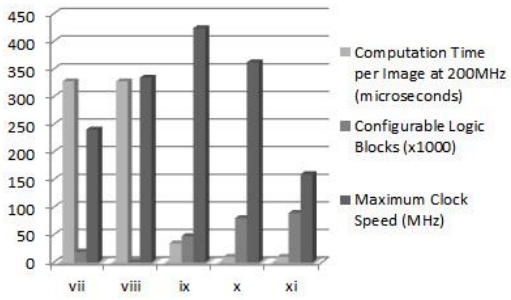


Figure 4. Area-time comparison of various hardware implementations of AHT and the regular HT on an FPGA

applications with area constraints, (ix) would provide a good tradeoff between area occupied and processing time, still being 12× faster than (viii). In applications where the angles for which HT is to be computed are known only at runtime, LUTs cannot be used. In such cases, CORDIC-based implementations could be used, with AHT implementation (xi) being 21× faster than the comparable regular HT implementation (vii).

While AHT implementations do occupy more on-chip area, this is justified by the lower area-time measure of AHT implementations. As shown by us in [1], the grid size can be adjusted to an optimal value, so as to minimize the area occupied. For a 256 × 256 image, a 10 × 10 grid is optimal. We have, however, in all our implementations used a 32 × 32 grid. As shown in [1], even though the area occupied is minimum for a 10 × 10 grid, the area-time cost continues to decrease with higher grid resolutions and applications not constrained by on-chip area would use finer grids to achieve faster execution. On lower-end devices, a grid size of 10 × 10 could be used to satisfy the area constraint.

## V. CONCLUSIONS

We have implemented AHT on a variety of computing platforms and have comprehensively evaluated its performance in these environments. It was shown that AHT provides significant benefits in computation time and area-time cost in both serial and parallel configurations, irrespective of the computing platform. In software, the effectiveness of the block-based nature of AHT on low-end GPUs and the scalability of AHT onto multiple CPU cores is demonstrated. We have also demonstrated the low computational complexity of AHT by comparing its serial configuration with the OpenCV implementation of HT on a processor-based platform. On reconfigurable hardware we proposed three different variations of AHT based on the application requirements. We demonstrated a CORDIC-based AHT implementation executes significantly faster than conventional CORDIC-based HT implementations. In applications where the angles for which HT is to be computed are known a priori, we presented two versions of AHT using LUTs instead of CORDIC, depending on the on-chip area available. Future work involves lowering the area cost without compromising on the parallelism offered by AHT.

## REFERENCES


[1] S. S. Sathyanarayana , R. K. Satzoda and T. Srikanthan "Exploiting inherent parallelisms for accelerating linear Hough transform," IEEE Trans. Image Process., vol. 18, no. 10, pp.2255 -2264 2009.

[2] R. O. Duda and P. E. Hart, "Use of the Hough transform to detect lines and curves in pictures," ACM Commun., vol. 15, no. 1, pp. 11–15, 1972.

[3] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in Proc. ACM/SIGDA 6th Int. Symp. FPGA, Feb. 1998, pp. 191–200.

[4] Suchitra, S., Satzoda, R. K., Srikanthan, T., "Accelerating CORDIC for hough transform," Integrated Circuits, ISIC '09. Proceedings of the 2009 12th International Symposium on , vol., no., pp.167,170, 14-16 Dec. 2009.

[5] Timmermann, D., Hahn, H., Hosticka, B.J., "Hough transform using Cordic method," Electronics Letters , vol.25, no.3, pp.205,206, 2 Feb. 1989

[6] S. M. Karabernou and F. Terranti, "Real-time FPGA implementation of Hough Transform using gradient and CORDIC algorithm," Journal of Image and Vision Computing, vol.23, pp. 1009 - 1017, July 2005.

[7] M.-Y. Chern and Y.-H. Lu, "Design and integration of parallel Hough transform chips for high-speed line detection," in Proc. 11th Int. Conf. Parallel and Distributed Systems, Jul. 2005, vol. 2, pp. 42–46.

[8] E. K. Jolly and M. Fleury, "Multi-sector algorithm for hardware acceleration of the general Hough transform," Image Vis. Comput., vol. 24, pp. 970–976, 2006.

[9] S. Mohanty and P. Hristov, "GPU accelerated Hough transform for high level trigger application", Proceedings of the DAE Symp. on Nucl. Phys. 57, 2012

[10] Gert-Jan van den Braak, Cedric Nugteren, Bart Mesman and Henk Corporaal, "Fast Hough transform on GPUs: exploration of algorithm trade-offs," Proceedings of the 13th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS'11), 2011

[11] TriCore Core Architecture User Manual v1.6