Image Processing Applications on a Low Power Highly Parallel SIMD Architecture

Amir Fijany
Italian Institute of Technology
Genova, Italy
amir.fijany@iit.it

Fouzhan Hosseini
Italian Institute of Technology
Genova, Italy
fouzhan.hosseini@iit.it

Abstract—In this paper, we present and discuss high performance implementation of a wide class of image processing applications on a low-power massively parallel SIMD architecture, the ClearSpeed CSX700. We present parallel implementation results for four classes of image processing applications: feature detection (Harris Corner Detector), stereo vision (a class of SSD like algorithms), model estimation (RANSAC), and object detection (based on Histogram of Oriented Gradient, HOG) on the CSX SIMD architecture. Our results indicate that this SIMD architecture is indeed a good candidate for achieving low-power supercomputing capability, as well as a rather satisfactory degree of flexibility for implementing various applications. We also compare our results, when applicable, with similar implementations on ASIC, FPGAs, and GPGPUs. This comparison cealrly demonstrates that we achieve a much better absolute computational performance than ASICs and FPGAs, with a better relative performance per watt. Compared with GPGPUs, we achieve similar (and for some cases better) computational performance but with a significantly better relative performance per watt. We show that, by designing appropriate effient parallel algorithms, this highly parallel SIMD architecture can represent an excellent candidate for space-borne applications wherein low-power, light weight, high performance computation is a major requirement.

TABLE OF CONTENTS

1. INTRODUCTION

Currently, we are interested in mobile robots and humanoids as an interesting and challenging example of emerging embedded computing applications. On one hand, in order to achieve a large degree of autonomy and intelligent behavior, these systems require a very significant computational capability to perform various tasks. On the other hand, they are severely limited in terms of size, weight, and particularly power consumption of their embedded computing system since they should carry their own power supply. Moreover, since these systems need to run various range of applications, their computing system should provide both flexibility and high performance.

Various image processing applications are widely used in mobile robots, for example, for 3D modeling of environment [1], obstacle avoidance [2], navigation [3], object tracking [4], etc. Conventional computing architectures, even fully exploiting their features, cannot provide adequate performance for real-time computation [3] and/or in terms of low power requirements. In fact, while conventional computing architectures provide flexibility, their limitation for this type of applications is twofold: first, their low computing power, second, their high power consumption.

Many aerospace applications share the same requirements, in terms of computational capability, and limitations, in terms of power consumption. In particular, they require high performance image processing capability for various applications. The NASA/JPL Mars Exploration Rovers (MERs) represent salient examples of low power mobile robots that rely on their on-board computing system to achieve local autonomy. Both Spirit and Opportunity rovers rely on stereo vision computation for depth map estimation for autonomous navigation [5], [6]. The rovers were designed to be, as much as possible, limited in terms of power consumption since they need to produce energy by using their solar panels. Also the on-board processor, the RAD6000, was designed to be very low power. The RAD6000 processor is able to compute one depth map each 30 seconds for images of resolution of 256x256 [5] while consuming 7W. We mentioned these two examples of Mars missions since they are severely limited in terms of energy while demanding a rather high computational capability. In fact, they represent the prime examples of the need for high computing power capabilities in very low power mobile robots. On the other hand, emerging mobile robot applications will be expected to achieve the same, if not more, level of local autonomy while relying on their embedded computing systems.

For image processing applications, in order to achieve better performance and/or lower power consumption, two main directions have been explored. In the first, ASICs and FP-

[1] 978-1-4244-7351-9/11/$26.00 ©2011 IEEE.
[2] IEEEAC Paper #1777, Version 3, Updated 26/10/2010.

GAs are designed and deployed for specific application to achieve better computational performance and much lower power consumption than conventional computing architectures. However, while ASIC and FPGA, due to their low power consumption, are very suitable for embedded applications, they suffer from lack of flexibility. Furthermore, the computational performance strongly depends on the level of parallelism offered by both the application and the hardware. Another direction is to use General Purpose Graphical Processing Units (GPGPUs). GPGUs can theroretically offer a significant computational power, due to their peak perfoprmance, but they are not suitable for embedded applications due to their rather large power consumption. Also, they offer some degree of flexibility and programmability for implementation of various applications. However, for many applications involving complex data structure and dependency, only a limited computational power of GPGPUs can be exploited.

Emerging highly parallel and low-power SIMD and MIMD architectures, for example ClearSpeed CSX [7] and Tilera [8], provide a unique opportunity to overcome the limitations of conventional computing architectures as well as ASICs, FPGAs, and GPGPus. These parallel architectures provide a much higher computing performance over conventional architectures while consuming significantly less power, resulting in significantly better overall performance in terms of GFLOPS or GOPS per watt. They also provide a large degree of flexibility and programmability. However, the main challenge in efficient exploitation of the capabilities of these emerging parallel architecture is the design of appropriate algorithm for the target application.

In this paper, we present and discuss high performance implementation of four classes of image processing applications on a highly parallel SIMD architecture, the ClearSpeed CSX architecture. This architecture has two cores, each with 96 Processor Elements (PEs), with a peak computing power of 96 GFOLPS, while consuming less than 9 Watts. We present parallel implementation results for four classes of image processing applications: feature detection (Harris Corner Detector), stereo vision (a class of SSD like algorithms), model estimation (RANSAC), and object detection (based on Histogram of Oriented Gradient, HOG) on the CSX SIMD architecture. While these considered applications have rather significantly different computational features in terms of granularity, data dependency, pattern of communication, etc., our implementation results clearly demonstrate that this architecture can provide excellent performance and flexibility in computing these applications. We discuss challenges in efficient algorithms design and programming for this architecture. We also compare our results, when applicable, with similar implementations on ASIC, FPGAs, and GPGPUs. This comparison cearly demonstrates that we achieve a much better absolute computational performance than ASICs and FPGAs, with a better relative performance per watt. Compared with GPGPUs, we achieve similar (and for some cases better) computational performance but with a significantly better relative performance per watt.

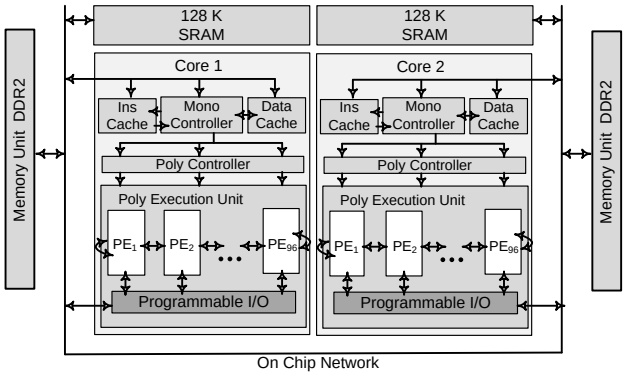

Figure 1. Simplified CSX Chip Architecture

This paper is organized as follows. In Section 2, we briefly discuss the CSX architecture and challenges for effiecient algorithms design. In Section 3, we discuss the four class of image processing applications and their parallel implementation on the CSX architecture. In Section 4, we present and discuss our parallel implementations results and compare them with other implementations. Finally, some concluding remarks are made in Section 5.

2. CSX Architecture and Algorithmic Challenges

Architecture Overview

In this section, we briefly review the ClearSpeed CSX 700 architecture with emphasis on some of its salient features that have been exploited in our implementations (see, for example, [7], [9] for more detailed discussion). As illustrated in Fig. 1, CSX700 has two similar cores, each core has a DDR2 memory interface and a 128KB SRAM, called external memory. Each core also has a standard, RISC-like, control unit, also called *mono execution unit*, which is coupled to a highly parallel SIMD architecture called *poly execution unit*.

Poly execution unit consists of 96 processing elements (PEs) and performs parallel computation (see Fig. 2). Each PE has a 128 bytes register file, 6KB of SRAM, high speed I/O channels to two adjacent PEs, as well as external I/O. It also includes an ALU, an integer multiply-accumulate (MAC) unit, and an IEEE 754 compliant floating point unit (FPU) with dual issue pipelined add and multiply, as well as support for floating point division and square root. The computational units within each PE can operate both in parallel and pipelined fashion. However, in order to better exploit these parallel and pipeline capabilities, specific instructions set called vector type operations should be used.

Poly execution unit includes a Programmable I/O (PIO) unit (Fig. 2) which is responsible for data transfer between external memory and PEs' memories, called poly memory. The

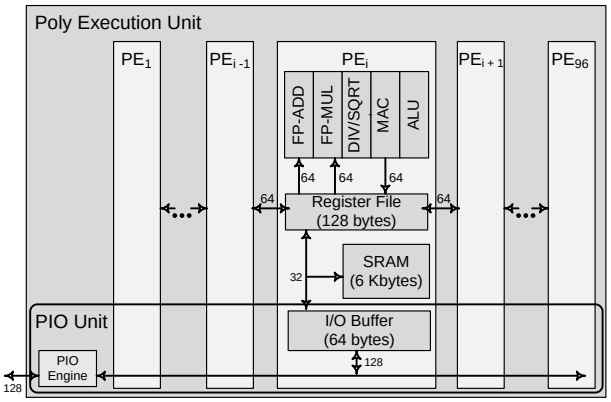

Figure 2. Simplified CSX Core Architecture

PIO unit serially transfers data form each of 96 PEs' memories to the external memory and vice versa. As shown in Fig. 2, the PIO unit consists of a PIO engine and 96 I/O buffers. Each I/O buffer is connected to the register file and memory of a PE. In fact, serial data transfer is performed to/from each PE's I/O buffer. It is important to note that this architecture enables the computational units and the PIO unit to work in parallel, thus enabling overlapping of communication with computation. This feature is fully exploited in our implementations to reduce I/O overhead.

Moreover, each PE is capable of communicating with its two neighboring PEs by using a dedicated bus called *swazzle path*. As shown in Fig. 2, swazzle path connects the register file of each PE to the register files of its left and right neighbors. Consequently, on each cycle, PEs are able to perform a register-to-register data transfer to either their left or right neighbor, while simultaneously receiving data from the other neighbor. The swazzle path can work in a pipeline fashion. And, in fact, there are some assembly instructions that allow swazzling of sizes up to 32 bytes much faster than calling the swazzle functions on multiple of 8-byte objects. The swazzle path provides the facility for parallel data communication among PEs.

Furthermore, the CSX architecture provides two mechanisms for global broadcasting of data. In the first mechanism, assuming that the data is already resident in the register file of the mono execution unit, it can then be transfered from mono register file to all PEs' register files using a single fast move instruction. Prior to this operation, data could have been transfered either from external memory (data cache) to mono register file or it could have been the result of some prior operations. In the second mechanism, global broadcasting can be performed by the PIO unit. For sending the same data to all PEs, i.e. global broadcasting of data, PIO unit performs one external memory read, and then distribute the data to all PE's I/O buffers [3]. The actual realistic cost of performing the PIO operations when data is in external memory and needs to be operated on that in poly registers also involves overhead for the load operations. Although the first mechanism is faster, it requires coding in assembly to get the best performance. Moreover, the benefit of using the PIO over move instruction by mono is that the PIO controller is a separate functional unit so the computational units can still be doing useful work while the PIO unit is transferring data.

Challenges for Efficient Algorithms Design

In order to fully exploit the features of the CSX architecture, any parallel algorithm design should consider the following specific issues.

Efficient use of PEs—The excellent performance per watt of the CSX architecture is achieved by integrating a large number of rather slow PEs. Therefore, any algorithm design should attempt to exploit a high degree of parallelism in the computation so that the 192 PEs are used as much as possible. Our proposed algorithms for the class of considered image processing applications indeed achieve a high degree of PEs utilization. In fact, as will be discussed in Section 3, for the multi-scale computation of HOG, such an efficient usage strategy can even require the solution of an optimization problem.

External Memory Communication Overhead—As mentioned before, PIO unit serially transfers data from external memory to each of 96 PEs' memories and vice versa. Here, care should be taken, otherwise communication could become a significant part of total running time. For example, as reported in [10] for an application on the CSX architecture, the external memory communications take 80% of total time. Such a significant communication overhead represents a major obstacle for achieving any optimal performance by using the CSX architecture. An alternative, when possible, is to reduce this overhead by overlapping communication with computation. We have extensively used such an overlapping of the computation with external memory communication in our algorithm design. For the CSX architecture, in order to fully utilize the underlying bus bandwidth, at each communication step the data size has to be at least 32 bytes, i.e. the time required to transfer 32 byte data or less is almost the same. For image processing applications wherein pixels are represented with 8 bit data, this means that at each communication step, 32 pixels could be transfered to each PE. Consequently, the overhead of external memory communication is not significant. However, we further improved the performance of implemented applications with a maximum overlapping of computation with communication.

PE's Memory Size—An important consideration for the CSX architecture is the size of PE's memory which is rather limited. Consequently, PEs might need to receive data from external memory or other PEs. Data transfer from external memory to PEs' memory (poly memory) is much more expensive than inter-PE communication via swazzling path.

[3] When same data is transfered to multiple PEs, the PIO unit attempts to minimize the number of external memory accesses. However, in practice, there is no guarantee that the minimum possible number will actually be achieved.

Vector Operations— The CSX700 has a clock cycle of $250MHz$[11]. Considering one add and one multiply floating point units working in parallel and generating one result per clock cycle, the peak performance of each PE is then 500 MFOPS, leading to a peak performance of 48 GFLOPS for one core and 96 GFLOPS for two cores (one chip). However, sequential (i.e., scalar) operations, wherein single add or multiply is performed, take 4 clock cycles to be performed [11]. This results to a sequential peak performance of 62.5 MFOPS for each PE, 8 GFLOPS for one core, and 12 GFLOPS for two cores (one chip). This indeed represents a drastic reduction in the peak, and hence, achievable performance. However, vector instructions which operate on sets of 4 data are executed much faster, e.g, vector add or multiply instructions take 4 cycles to be completed [11]. Therefore, vector instructions allow greater throughput for operations. In addition to the use of vector instruction, we have also used assembly coding to further improve the performance.

Compiler—The code generated by compiler may not be optimized. Therefore, in order to make sure the best performance is achieved, we have also written part of our codes in assembly language of the CSX.

3. TARGET APPLICATIONS

we have considered high performance implementation of four classes of image processing applications: feature detection (Harris Corner Detector), stereo vision (a class of SSD like algorithms), model estimation (RANSAC), and object detection (based on Histogram of Oriented Gradient, HOG) on the CSX architecture. In this section, we briefly describe the algorithms and discuss their parallel implementation.

Harris Corner Detector

Harris Corner Detector (HCD) [12] is a popular corner detector due to its invariance to rotation, scale, illumination variation and image noises. To detect corners in a given image, the HCD algorithm [12] proceeds as the following. Let $I(x, y)$ denote the intensity of a pixel located at row x and column y of the image.

1. For each pixel (x, y) in the input image compute the elements of the Harris matrix $G = \begin{bmatrix} g_{xx} & g_{xy} \\ g_{xy} & g_{yy} \end{bmatrix}$ as follows:

$$g_{xx} = \left(\frac{\partial I}{\partial x}\right)^2 \otimes w$$
$$g_{xy} = \left(\frac{\partial I}{\partial x}\frac{\partial I}{\partial y}\right) \otimes w$$
$$g_{yy} = \left(\frac{\partial I}{\partial y}\right)^2 \otimes w, \quad (1)$$

where $\otimes$ denotes convolution operator and w is the Gaussian filter.

2. For all pixel (x, y), compute Harris' criterion:

$$c(x, y) = \det(G) - k(trace(G))^2 \quad (2)$$

where $\det(G) = g_{xx}.g_{yy} - g_{xy}^2$, k is a constant which should be determined empirically, and $trace(G) = g_{xx} + g_{yy}$.

3. Choose a threshold τ empirically, and set all $c(x, y)$ which are below τ to zero.

4. Non-maximum suppression, i.e. extract points (x, y), which have the maximum $c(x, y)$ in a window neighborhood. These points represents the corners.

Fast implementations of HCD on various architectures have been considered in the literature including Application Specific Integrated Circuit (ASIC) [13], Field Programmable Gate Array (FPGA) [14], Graphics Processing Units (GPU) [15], and Cell processor [16]. A performance comparison of these implementations is given in Section 4.

Parallel Implementation on the CSX Architecture. Considering the SIMD architecture of CSX, we have employed data parallel model of computation. The first issue in any data parallel implementation is an efficient data distribution, i.e., assigning data to PEs, scheme. Having an image and an array of PEs, various data distributions schemes could be considered. The most obvious schemes are row (column)-stripe distribution, block distribution, and row (column)-cyclic distribution. Let p indicate the number of PEs. Also let c and r denote the number of image columns and rows, respectively. In block distribution scheme, the image is divided into $p = d * s$ blocks, with each block having c/d columns and r/s rows. The first block is assigned to the first PE, the second one to the second PE, and so on. In row-strip distribution, the first r/p rows are assigned to the first PE, the second r/p rows are assigned to the second PE, and so on. Finally in row-cyclic scheme, the first row is assigned to the first PE, the second row to the second PE, and so on.

An important consideration for implementing HCD on the CSX architecture is the size of PE's memory which is rather limited. For the CSX architecture, various data distributions should be compared in terms of the following parameters: (a) required memory space for each PE; (b) redundant external memory communication; and (c) inter-PE communication time. We have analyzed the computation of HCD and have shown that row-cyclic data distribution scheme is the most efficient for parallel implementation of HCD on the CSX architecture [17]. In fact, Row-cyclic distribution needs less poly memory space and no redundant external memory communication.

As mentioned in Section 2, each CSX core includes 96 PEs. To apply the row-cyclic distribution scheme for computation, the input images are divided into groups of 96 rows and the computation is performed in several iterations (sweeps), denoted as outer loop iteration. In each iteration, operations are performed on a group of 96 rows. To handle boundary conditions, two consecutive iterations are overlapped.

Also to overcome the overhead of external memory communication, communication and computation overlapping is

greatly exploited in our implementation. To achieve maximum overlapping, each row is divided into a set of segments of size m. The computation for each row is then performed in several iterations (sweeps), denoted as inner loop iterations. In each inner loop iteration, the computation is performed for a segment of data, i.e., m pixels.

SSD-Based Stereo Vision

The purpose of stereo vision computation is to estimate the depth map of a 3D environment from two images captured at the same time and with slightly different viewpoints. Stereo vision has been extensively investigated and a great variety of algorithms have been developed [18]. The SSD algorithm [18] is a window-based approach to obtain the disparity map on a pair of rectified stereo images. To describe the algorithm, let $I_R(i,j)$ and $I_L(i,j)$ denote the intensity of pixels located at row i and column j in the right and left images, respectively. The input parameters of the algorithm are ω, the window size, and β, the maximum disparity. Assuming the right image as reference, the disparity for each pixel (i,j) in the right image is calculated as follow:

- Consider a window centered at (i,j) in the right image
- Consider a window centered at $(i,j+k)$ in the left image where $j \leq k < j+\beta$
- Calculate convolution of the windows in the left and right images as

$$S(i,j,k) = \sum_{l=i-\frac{w-1}{2}}^{i+\frac{w-1}{2}} \sum_{m=j-\frac{w-1}{2}}^{j+\frac{w-1}{2}} \Big[I_R(l,m) - I_L(l,m+k)\Big]^2 \tag{3}$$

- The pixel that minimizes $S(i,j,k)$ is the best match. So,

$$k^* = \arg \min_{j \leq k < j+\beta} S(i,j,k), \tag{4}$$
$$d(i,j) = k^*$$

Briefly, the SSD algorithm consists of the following three steps:

1. Calculating the squared differences of intensity values for a given disparity
2. Summing the squared differences over square windows
3. Finding the pixels with the minimum sum of squared differences.

An extensive overview of stereo vision algorithms, with emphasis on application for intelligent vehicles, is presented in [3]. The results in [3] demonstrate that using a local method such as Sum of Squared Difference (SSD) algorithm along with a robust error rejection scheme, such as left-right check (i.e., using both images as reference) and multiple window computation, can lead to the best results. However, based on the results reported in [3] and [19], the conventional architectures, even fully exploiting their features, cannot provide adequate performance for real-time implementation of particularly more advance and hence more accurate stereo vision algorithms. For example, the implementation results reported in [3] are far from achieving a real time performance even for a 512x512 image.

In order to achieve real-time performance, a variety of approaches for implementation of stereo vision on special-purpose and high performance architectures have been proposed [20], [21], [22], [23]. Most of the special-purpose architectures proposed for fast and real-time computation of stereo vision have been focused on the implementation of SSD algorithm and for, most cases, rather small image sizes. However, the more accurate is the depth estimation, the greater is the computational complexity of the algorithm. Using a left-right check scheme increases the computation by a factor of two, while using multiple window schemes not only increases the computational cost but also leads to more complex data communication patterns, making the design of efficient special-purpose architectures more difficult. The computation cost also increases with the image size. In fact, it seems that efficient implementation for emerging larger image sizes, such as HDTV with a resolution of 1280×720, has not been extensively considered in the literature.

Parallel Implementation on CSX Architecture. We have analyzed the computation of SSD algorithm and showed that row-cyclic data distribution scheme is the most efficient for implementing SSD algorithm (and also for the two variations of SSD, multiple window and left-right extensions) on the CSX architecture [24], [25]. In fact, Row-cyclic distribution needs less poly memory space and no redundant external memory communication. The implementation of SSD algorithms on the CSX architecture consists of outer loop and inner loop iterations as described in HCD implementation.

RANSAC

The RANSAC (RANdom SAmple Consensus) algorithm, originally developed by Fishler and Bolles [26], has become a fundamental tool for robust model estimation in computer vision and image processing applications [27].

RANSAC is an iterative method to estimate parameters of a certain mathematical model from a set of data which may contain a large number of outliers. Each iteration of RANSAC consists of the following two steps.

- Model Generation Step: A minimal sample set (MSS) is randomly selected from the dataset. Cardinality of MSS is smallest sufficient number of data to determine model parameters. Then, parameters of the model are computed, using only MSS elements .
- Model Verification Step: RANSAC determines the set of data in entire dataset which are consistent with the model and parameters estimated from MSS in previous step. This set of data is called consensus set (CS).

These steps are performed iteratively until the probability of finding a better CS drops below a certain threshold and

RANSAC terminates.

To describe RANSAC more formally, assume that dataset, consisting of N elements, is indicated by $\mathcal{D} = \{\mathbf{d^1}, \mathbf{d^2}, \cdots, \mathbf{d^N}\}$ and $\boldsymbol{\theta}$ denotes the parameter vector. Let $\mathcal{S}$ denote a selected MSS, and $err(\boldsymbol{\theta}, \mathbf{d^i})$ be an appropriate function which indicates the error of fitting datum $\mathbf{d^i}$ in the model with parameter vector $\boldsymbol{\theta}$. RANSAC algorithm proceeds as following. First, $\mathcal{S}$ is randomly selected from dataset $\mathcal{D}$, and the model parameters $\boldsymbol{\theta}$ are computed based on the elements in $\mathcal{S}$ (model generation step). In the next step, RANSAC checks which elements in $\mathcal{D}$ fit in the model. Each datum $\mathbf{d^i}$ is considered to fit the model, if its fitting error, $err(\boldsymbol{\theta}, \mathbf{d^i})$, is less than a threshold δ. If this is the case, then the datum is added to the consensus set, CS (model verification step). After that, the CS is compared with the best consensus set CS^* obtained so far. If CS is ranked better than CS^*, best consensus set and best model parameters are updated.

For many applications a real-time implementation of RANSAC is indeed desirable. However, its computational complexity represents a major obstacle for achieving such a real-time performance. The computational complexity of RANSAC is a function of the number of required iterations, i.e., the number of generated hypothetical models, and the size of data set. In fact, RANSAC can often find the correct model even for high levels of outliers [28]. However, the number of hypothetical models required to achieve such an exact model increases exponentially, leading to a substantial computational cost [28]. Consequently, there has been significant effort in improving the performance of RANSAC by either reducing the number of models, e.g. [29], [30], [31], or by reducing the size of data set for model evaluation [27], [32], [33].

An efficient alternative to improve the performance of RANSAC is to speed up the computation by exploiting parallelism. To our knowledge, such a parallel implementation has not been extensively and rigorously considered in the literature. In fact, it seems that the only reported work on parallel implementation of RANSAC is by Iser et al. [34] wherein a very limited parallelism has been exploited. In [34] the implementation of pRANSAM algorithm, a limited parallelization of the RANSAM algorithm [35], on an Intel multi-processor chip has been considered.

Parallel Implementation on the CSX Architecture. One can consider a rather straightforward parallel implementation of RANSAC by exploiting parallelism in each iteration. Note that at each iteration, a same model is evaluated for all the elements of the data set. This represents a data parallel computation since the evaluation for all the elements of the data set can be performed in parallel. However, this approach has certain limitations. First, model estimation is performed serially, i.e. PEs are idle during model estimation. Also, the utilization of PEs depends on data set size.

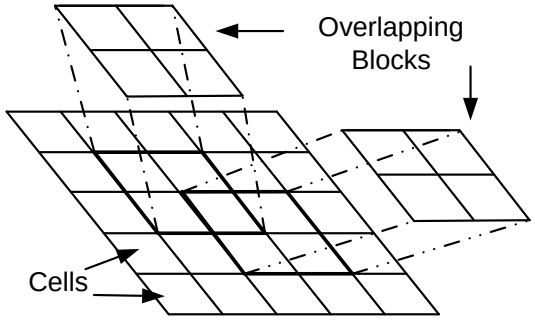

Figure 3. Division of the Image in Cells and Blocks for Calculating HOG Descriptor

Our parallel implementation of RANSAC denoted as P-RANSAC can be considered as a multi-stage process wherein, at each stage, a large number of models are generated and evaluated in parallel. The checking is then performed at the end of each stage to determine whether more stages are needed. This approach is motivated by the simple observation that the iterations of RANSAC are, to a large degree, independent and can be performed in parallel.

Note that, while this strategy provides a massive degree of parallelism in the computation, all the processors need to have access to the whole set of data. On the CSX architecture this can be efficiently achieved by using the global broadcasting capability since the elements of the data set can be broadcast to all PEs. Interestingly, the PEs do not even need to store the data set. In fact, for many practical applications, the size of the data set can be even bigger than the available memory space in the PEs! As was discussed before, this global broadcasting can be fully overlapped with the PEs computation and thus do not at all degrade the performance. The main advantage of this strategy is that it provides a massive degree of parallelism with a minimum overhead.

Histogram of Oriented Gradient

Histogram of Oriented Gradient (HOG) descriptor [36] is being widely used in computer vision for object detection, due to its excellent performance.

To calculate HOG descriptor, the image is first divided into small spatial regions called *cell*s and the histogram of gradient orientation is calculated for each cell. Each cell is considered as a 2D array of $m \times m$ pixels. Then, histograms are normalized over groups of cells, called *block*s. Every block is a 2D array of $n \times n$ cells. There is overlapping among the blocks since each cell participates in several blocks (Fig. 3).

Calculation of HOG descriptor consists of three steps: first, the magnitude and the orientation of gradient is calculated for each pixel in the image. To see this, let $I(x, y)$ be the intensity of pixel located at row x and column y. Then, the magnitude of gradient, $m(x, y)$, and the orientation of gradient, $\theta(x, y)$

are calculated by:

$$m(x,y) = \sqrt{I_x(x,y)^2 + I_y(x,y)^2} \quad (5)$$

$$\theta(x,y) = \arctan \frac{I_y(x,y)}{I_x(x,y)}, \quad (6)$$

where $I_x(.,.)$ and $I_y(.,.)$ denote the partial derivation of I in direction x and y, respectively.

Second, the histograms are calculated for each cell. To calculate histograms, the orientation bin are evenly spaced over $0^\circ - 180^\circ$ or $0^\circ - 360^\circ$ into q bins. Each pixel calculates a vote for the histogram bin based on the orientation of gradient on that pixel, $\theta(x,y)$. The vote is also a function of gradient magnitude at that pixel, $m(x,y)$. Also to improve performance, the contribution of each pixel in cell histogram is weighted by a Gaussian function centered in the middle of the block. Moreover, weighted votes are interpolated trilinearly between the neighboring bin centers in both orientation and position, to reduce aliasing effect. For more details on interpolation see [37].

Finally, for each block, the histogram of its cells are combined and normalized to form the HOG descriptor of that block. For human detection *L2-norm* is used.

To detect objects, a sliding window based algorithm is used. First, a detection window is defined as a grid of blocks. The detection windows are overlapping since a given block can belong to several detection windows. The descriptor of the detection window is then obtained by combining the vectors of HOG descriptors of its blocks. For object detection, the detection window is scanned across the image at all positions and scales and the resulting descriptors are fed into a pretrained linear SVM classifier to score each descriptor.

Dalal and Triggs [36] have shown that HOG descriptors outperform existing descriptors for human detection. However, a major bottleneck in real-time applications of HOG descriptor is its computational complexity. In fact, in the first original implementation of the HOG descriptor on a conventional computer only a performance of 1 frame per second (fps), for a rather small image of 320x240 resolution and with a rather small number of detection window of 800, could be achieved.

Cao et al. [38] have studied implementation of HOG-based human detector on FPGA. However, they considered some modifications and simplifications to the original algorithm since HOG descriptors are not suitable for FPGA implementation due to their rather complex communication and data dependency patterns. Implementation of HOG descriptor has also been considered on General Purpose Graphic Processing Units (GPGPU) [39], [40]. The comparison of the results are given in Section 4.

Parallel Implementation on the CSX Architecture For parallel computation of HOG-based object detection, the challenges are complex data dependency pattern, varying granularity (pixel, cell, block, and detection window), and multi-scale computation. Multi-scale computation represents repeated computation of the same problem but with reducing size. Finding a mapping of computation on a SIMD architecture, which is efficient for various problem sizes, is challenging.

The computation for object detection by using HOG descriptor can be analyzed at four levels of operations: (a) pixel level, (b) cell level, (c) block level, and (d) detection level. Thus, the computation involves several level of operations with different granularity. Therefore, the first critical decision to achieve optimal performance is the choice of grain size. For our parallel implementation on the CSX SIMD architecture, we have shown that a choice of block as the main computation unit is the most efficient in terms of reducing the redundancy in the computation and communication [41].

With a block level granularity, the input image is subdivided into blocks and PEs are assigned to compute the HOG descriptor of blocks. Since, the number of blocks is greater than the number of PEs (96), each PE has to calculate the HOG descriptor of several blocks. Hence, the next decision is to determine the assignment of the blocks to PEs. Since HOG descriptors of blocks are used as the input for the detection level, this assignment also affects the computation at the detection level. We have shown that the optimal scheme for assignment of blocks to PEs is column-wise scheme, i.e. the blocks in the first m columns are assigned to the first PE, the blocks in the second m columns are assigned to the second PE, and so on [41]. This scheme of assigning blocks to PEs enable an efficient implementation of detection level computation.

Therefore, the number of required PEs to perform HOG calculation for one scale of image is determined by number of image columns. Consequently, if the number of required PEs is less than 96, some PEs will be idle. Moreover, the computation time is determined by number of image rows. Consequently, the computation of various scales can be considered as a set of tasks with different computation time and required execution recourse (PEs).

So far we have discussed the parallel implementation of HOG-based object detection on one scale of image. The same approach can be applied for all scales in several rounds where in each round the computation is performed for one scale of image. However, such a straightforward approach would be inefficient since the number of idle PEs increases as the scale of the image decreases. A consequence of our parallelization strategy is that regular operations are performed for all blocks by all PEs. Therefore, it is possible to perform the computation on several scales of input image simultaneously in each round to reduce the number of idle PEs and hence the total computation time. The key issue is then to find an optimal scheme for mapping the computations of various scales on the fixed number of PEs to minimize the total computation

time. This optimization problem is indeed equivalent to the solution of the *strip packing problem*.

Strip packing problem consists of packing a set of rectangular items of width at most W on a strip of fixed width W and of infinite height (see, for example, [42]). The items may neither overlap nor be rotated. The objective is to minimize the height used.

Since each CSX700 core has 96 PEs, it can be considered as a strip of width 96. Also, as shown in Fig. 4(a), the computation for scale s of input image can be considered as a rectangle item of width P_s and height I_s, where I_s and P_s denote the number of iterations and the number of PEs required for computation of scale s, respectively. The objective is to minimize the computation time which could be considered as the height used to place all the rectangles (i.e, the computation of all different scales of image) into a strip (one CSX700 core).

The strip packing problem is NP-Hard [42]. A class of commonly used heuristic algorithms for strip packing problems are level algorithms. We have used a best-fit decreasing height (BFDH) heuristic to solve the mapping problem [41]. An example of the packing scheme produced by the BFDH algorithm is illustrated in Fig. 4(b). In this example, computation is performed on 28 scales of the input image. Each level in the produced packing represents one round of computation, i.e. all scales of input image which are placed in one level are computed in one round.

It should be emphasized that the computations of rounds are totally independent and hence they can be performed in any order. If more than one CSX700 is used, rounds are divided into groups, and each group is assigned to one core. Number of groups is equal to number of available cores.

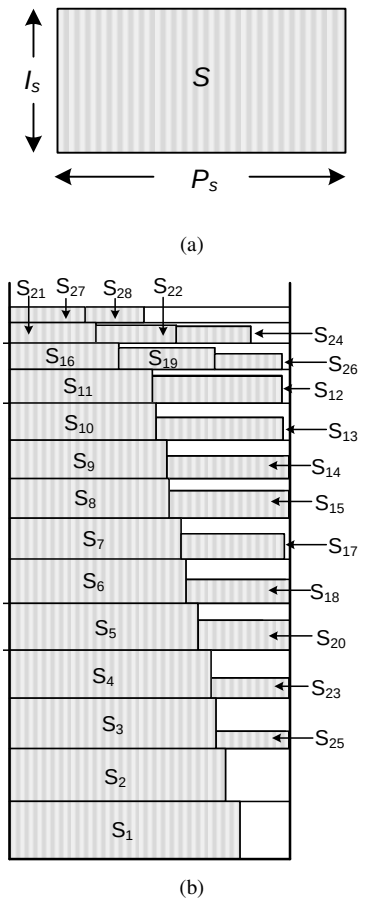

Figure 4. HOG-based Object Detection (a) Computation for each scale of image can be presented by a rectangular of width P_s (number of required PEs) and Height I_S (computation time) (b) An Example of Packing Produced by the BFDH Algorithm

4. RESULTS AND COMPARISON

In this section, the performance of four implemented applications discussed in Section 3 on CSX architecture are presented and compared with the other works in literature.

Harris Corner Detector

We have implemented the following variants of HCD algorithm on the CSX700 architecture: $HCD_{3\times3}$ and $HCD_{5\times5}$ by using a 3×3 and 5×5 Gaussian kernel, respectively. Since our proposed parallel approach provides flexibility, it can be easily applied to images with different sizes, and to various sizes of Gaussian filter or non-maximum suppression window. The performance of implemented algorithms in terms of latency, frame per second (fps), and sustained GFLOPS for different image resolutions are summarized in Table 1. As Table 1 shows, for all tested image resolutions, even for resolution of 1280x720, our implementation is much faster than real-time.

Table 2 compares our implementation results with those reported in the literature. As can be seen, our approach provides much better performance in terms of latency or frame per second while providing a high degree of flexibility in terms of problem size and parameters.

SSD-Based Stereo Vision

We have implemented following stereo vision algorithms on the CSX architecture: (a) SSD: basic SSD (b) SSD_MV5: SSD with 5 windows (c) SSD_LR: SSD with left-right check

Table 3 presents the performance of implemented stereo vision algorithms in terms of latency, fps, and sustained GFLOPS for images of 640x480 and 1280x720 resolutions. As shown in this table, we achieve real-time, and for most cases even faster than real-time, performance for all considered problem instances except for 1280x720 images with disparity of 32, for which only SSD algorithm achieves the real-time performance.

Another important observation is that, as our practical results demonstrate, the computation time of different algorithms

Table 1. Performance of HCD on CSX700 Architecture Using 3×3 and 5×5 Gaussian Filter

Image Resolution	Latency (ms)		fps		Sustained GFLOPS	
	$HCD_{3\times3}$	$HCD_{5\times5}$	$HCD_{3\times3}$	$HCD_{5\times5}$	$HCD_{3\times3}$	$HCD_{5\times5}$
128x128	.165	.224	6060	4464	3.97	4.68
352x288	.8	1.22	1250	819	5.06	5.31
512x512	1.74	2.63	574	380	6.02	6.37
640x480	2.15	3.28	465	304	5.71	5.99
1280x720	7.04	10.89	142	91	5.23	5.41

Table 2. Comparison of HCD Implementation on CSX With Other Implementations in the Literature

Image Resolution	fps reported in [ref]	fps achieved by our approach
128x128	1367 (ASIC) [13]	4464
352x288	60 (FPGA) [14]	819
640x480	99 (GPU) [15]	304

Table 4. Comparison of Stereo Vision implementation on CSX and Conventional Architecture

Algorithm	MPDs reported in [3]	MPDs reported in [25]
SSD	143	1032
SSD_MW5	75	442
SSD_LR	114	501

closely correlate with their computation cost in terms of the disparity range. That is, the computation time almost linearly increases with the disparity range, e.g. the computation times almost doubles for a factor of 2 increase in the disparity range. This would enable the prediction of the performance for any disparity. This close correlation is due to the fact that the disparity range only directly affects the amount of the computation performed by each PE.

Table 4 compares the performance of stereo vision algorithms implemented on CSX architecture and on conventional architecture by exploiting SSE instructions [3]. The mega-pixel disparities per second (MPDs) are obtained by multiplying 4 values : number of image columns, number of image rows, maximum disparity, and fps.

RANSAC

We have applied the P-RANSAC for the example of estimating parameters of a homography transform. Homography is a linear transformation in projective space which relates two images of a planner scene taken from different view by pinhole camera. In the field of computer vision, homography transformation has many applications, such as image rectification, image registration, and structure form motion. We refer the reader to [43] for more details on using RANSAC to estimate homography transform.

Table 5. Performance in One Stage: for Homography Estimation Number of Data=1024, Number of Model=192

Steps	Latency (ms)	Sustained GFLOPS
Model Generation	2.053	4.22
Model Verification	.2286	20.21
Memory Overhead	.00052	-
Total	2.282	5.82

Table 6. Performance of HOG-base Human detection on One core and Two Core of CSX

	Latency (ms)	fps	Sustained GFLOPS
1 core	292.03	3.4	1.51
2 core	146.23	6.8	3

The performance of P-RANSAC in terms of latency and sustained GFLOPS for estimating homography transform are illustrated in Table 5.

As was discussed in Section 2, the CSX architecture allows overlapping the computation of each PE with the data transfer to and from its memory. We have extensively exploited this capability in our parallel implementation of RANSAC. As can be seen from table 5, the external memory communication overhead takes only .52 μs, representing just .02% of the total computation time.

Histogram of Oriented Gradient

We have implemented our proposed parallel algorithm for HOG-based human detection on the CSX700 SIMD architecture. By using both cores of the CSX700 processor, we have achieved a performance of 6.8 fps. Table 6 compares our implementation of one and two cores of CSX700. Our results illustrates that using both cores of CSX700, we achieved a near perfect speedup of two.

Table 3. Performance of implemented Stereo Vision algorithms on CSX700 Architecture

Image Resolution	Algorithm	Latency (ms)		fps		GFLOPS	
		$\beta = 16$	$\beta = 32$	$\beta = 16$	$\beta = 32$	$\beta = 16$	$\beta = 32$
640x480	SSD	5.57	10.56	179	94	6.17	6.51
	SSD_MW5	12.5	24.51	80	40	4.71	4.81
	SSD_LR	11.05	21.07	90	47	3.58	3.74
1280x720	SSD	14.72	28.04	67	35	7.01	7.36
	SSD_MW5	33.2	65.18	30	15	5.32	5.42
	SSD_LR	29.51	56.58	33	17	4.02	4.18

Table 7. Comparison of HOG implementation on CSX Architecture with Other Implementations in the Literature

	Latency (ms)	Peak Performance (GFLOPS)	fps / watt
CSX [41]	146.23	96	.75
GPU [39]	99	384	.06
GPU [40]	67	1788.48	.05

Table 7 compares our implementation of HOG-based human detection on CSX architecture with those reported in the literature by using GPGPUs. Although, our implementation, by using only one CSX architecture, is slower than those on the GPGPUs in terms of computation time, but, in terms of power consumption and particularly fps per watt, it achieves a much better performance. To achieve better performance in terms of computation times, more CSX700 processors can be used. As discussed in Section 3, the multi-scale computation can be divided among more cores, i.e. by using multiple CSX700, enabling almost a linear speedup. For example, by using 4 CSX boards a performance of about 25 fps can be achieved while consuming 36 watts.

5. CONCLUSION

We presented and discussed high performance implementation of four classes of image processing applications on the highly parallel CSX SIMD architecture. We presented parallel implementation results for feature detection (Harris Corner Detector), stereo vision (a class of SSD like algorithms), model estimation (RANSAC), and object detection (based on Histogram of Oriented Gradient, HOG) on the CSX SIMD architecture. As discussed, these considered applications have rather significantly different computational features in terms of granularity, data dependency, pattern of communication, etc.. For example, our parallel implementation for HCD and SSD represents a rather fine grain computation (parallelism is exploited at the pixel level) while for HOG is a medium grain computation (parallelism is exploited at the block level) and for RANSAC is rather coarse grain (parallelism is exploited at model generation and validation level). Furthermore, they involve different pattern of data communication, including global communication for RANSAC. Our implementation results clearly demonstrate that this architecture can provide excellent performance and flexibility in computing these applications.

Currently there is a lot emphasis on the use of FPGA and GPGPUs for image processing applications. However, the comparison of our implementation results with the similar ones on the ASIC and FPGA for HCD and SSD computation clearly demonstrated that we achieved a much better absolute computational performance than ASICs and FPGAs, with a better relative performance per watt. Compared with GPGPU, we achieved a lower computational performance for HOG but much higher for HCD computation while for both cases we achieved a significantly better relative performance per watt. We believe that such results can indeed further motivate the investigation and application of emerging highly parallel, low power, SIMD and MIMD architectures for future aerospace applications.

REFERENCES

[1] S. Fleck, F. Busch, P. Biber, W. Straßer, and H. Andreasson, "Omnidirectional 3d modeling on a mobile robot using graph cuts," in *IEEE International Conference on Robotics and Automation (ICRA'05),*, 2006, pp. 1748–1754.

[2] L. Nalpantidis, I. Kostavelis, and A. Gasteratos, "Stereovision-based algorithm for obstacle avoidance," in *2nd International Conference on Intelligent Robotics and Applications (ICIRA '09)*, 2009, pp. 195–204.

[3] W. van der Mark and D. M. Gavrila, "Real-time dense stereo for intelligent vehicles," *IEEE Trans. on Intelligent Transport System*, vol. 7, no. 1, pp. 38–50, 2006.

[4] F. Tang, M. Harville, H. Tao, and I. Robinson, "Fusion of local appearance with stereo depth for object tracking," in *Computer Vision and Pattern Recognition Workshops(CVPRW'08)*, 2008, pp. 142–149.

[5] L. Matthies, M. Maimone, A. Johnson, Y. Cheng, R. Willson, C. Villalpando, S. Goldberg, A. Huertas, A. Stein, and A. Angelova, "Computer vision on mars,"

Int. J. Comput. Vision, vol. 75, no. 1, pp. 67–92, 2007.

[6] M. Maimone, A. Johnson, Y. Cheng, R. Willson, and L. Matthies, "Autonomous navigation results from the mars exploration rover (mer) mission," in *9th Int. Symp. on Experimental Robotics(ISER)*, June 2004.

[7] *Clearspeed Whitepaper: CSX Processor Architecture*, ClearSpeed, www.clearspeed.com, 2007.

[8] http://www.tilera.com/.

[9] *CSX600 Hardware Programming Manual*, ClearSpeed, www.clearspeed.com, Jan 2008, document No. 06-RM-1305 Revision: 1.A.

[10] V. Heuveline and J.-P. Weiβ, "Lattice boltzmann methods on the clearspeed advance^{TM} accelerator board," *The European Physical Journal-Special Topics*, vol. 171, no. 1, pp. 31–36, 2009.

[11] *CSX600/CSX700 Instruction Set Reference Manual*, ClearSpeed, www.clearspeed.com, August 2008, 06-RM-1137 Revision: 4.A.

[12] C. Harris and M. Stephens, "A combined corner and edge detector," in *4th Alvey Vision Conference*, 1988, pp. 147–151.

[13] C.-C. Cheng, C.-H. Lin, C.-T. Li, S. C. Chang, and L.-G. Chen, "iVisual: an intelligent visual sensor SoC with 2790fps CMOS image sensor and 205GOPS/W vision processor," in *45th annual Design Automation Conference(DAC '08)*, 2008, pp. 90–95.

[14] B. Dietrich, "Design and implementation of an FPGA-based stereo vision system for the EyeBot M6," University of Western Australia, 2009.

[15] L. Teixeira, W. Celes, and M. Gattass, "Accelerated corner-detector algorithms," in *19th British Machine Vision Conference(BMVC '08)*, 2008, pp. 625–634.

[16] T. Saidani, L. Lacassagne, S. Bouaziz, and T. M. Khan, "Parallelization strategies for the points of interests algorithm on the cell processor," in *5th International symposium on Parallel and Distributed Processing and Applications (ISPA'07)*, 2007, pp. 104–112.

[17] F. Hosseini, A. Fijany, and J.-G. Fontaine, "Highly parallel implementation of harris corner detector on CSX SIMD architecture," in *4th Workshop on Highly Parallel Processing on a Chip (HPPC'10) in conjunction with Euro-par*, 2010.

[18] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, pp. 7–42, 2001.

[19] L. Di Stefano, M. Marchionni, and S. Mattoccia, "A pc-based real-time stereo vision system," *Int. Journal of Machine Graphics & Vision*, vol. 13, no. 3, pp. 197–220, 2004.

[20] N. Chang, T. Lin, T. Tsai, Y. Tseng, and T. Chang, "Real-time dsp implementation on local stereo matching," in *IEEE International Conference on Multimedia and Expo*, 2007, pp. 2090–2093.

[21] Y. Jia, X. Zhang, M. Li, and L. An, "A miniature stereo vision machine MSVM-III for dense disparity mapping," in *17th International Conference on Pattern RecognitionICPR'04*, vol. 1, 2004, pp. 728–731.

[22] J. Woodfill, G. Gordon, and R. Buck, "Tyzx deepsea high speed stereo vision system," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2004, pp. 41–45.

[23] F. Gurkaynak, M. Kuhn, S. Moser, O. Isler, A. Burg, N. Felber, H. Kaeslin, and W. Fichtner, "Efficient ASIC implementation of a real-time depth mapping stereo vision system," in *PROCEEDINGS OF THE IEEE MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS*, vol. 46, no. 3, 2003, p. 1478.

[24] F. Hosseini, A. Fijany, S. Safari, R. Chellali, and J.-G. Fontaine, "Real-time parallel implementation of ssd stereo vision algorithm on CSX SIMD architecture," in *5th International Symposium on Advances in Visual Computing (ISVC '09)*, 2009, pp. 808–818.

[25] F. Hosseini, A. Fijany, S. Safari, and R. Chellali, "Fast implementation of dense stereo vision algorithms on a highly parallel SIMD architecture," Submitted to Real-Time Image Processing.

[26] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[27] O. Chum and J. Matas, "Randomized RANSAC with $T_{d,d}$ test," in *Proc. British Machine Vision Conference (BMVC'02)*, Sep. 2002, pp. 448–457.

[28] R. Raguram, J.-M. Frahm, and M. Pollefeys, "A comparative analysis of RANSAC techniques leading to adaptive real-time random sample consensus," in *Proc. of the 10th European Conference on Computer Vision (ECCV '08)*, 2008, pp. 500–513.

[29] O. Chum and J. Matas, "Matching with PROSAC - progressive sample consensus," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, June 2005, pp. 220–226.

[30] B. J. Tordoff and D. W. Murray, "Guided-MLESAC: Faster image transform estimation by using matching priors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 10, pp. 1523–1535, 2005.

[31] D. R. Myatt, P. H. S. Torr, S. J. Nasuto, J. M. Bishop, and R. Craddock, "NAPSAC: High noise, high dimensional robust estimation," in *Proc. British Machine Vision Conference (BMVC'02)*, Sep 2002, pp. 458–467.

[32] D. Nistér, "Preemptive RANSAC for live structure and motion estimation," *Mach. Vision Appl.*, vol. 16, no. 5, pp. 321–329, 2005.

[33] J. Matas and O. Chum, "Randomized RANSAC with

sequential probability ratio test," in *Proc. IEEE International Conference on Computer Vision (ICCV'05)*, Oct 2005, pp. 1727–1732.

[34] R. Iser, D. Kubus, and F. M. Wahl, "An efficient parallel approach to random sample matching (pRANSAM)," in *Proc. International Conference of Robotics and Automation(ICRA'09)*, May 2009, pp. 1199–1206.

[35] S. Winkelbach, S. Molkenstruck, and F. M. Wahl, "Low-cost laser range scanner and fast surface registration approach," in *28th Annual Symposium of the German Association for Pattern Recognition (DAGM'06)*, Sep 2006, pp. 718–728.

[36] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, Jun 2005, pp. 886–893.

[37] N. DALAL, "Finding people in images and videos," Ph.D. dissertation, National Polytechnic Institute of Grenoble, July 2006.

[38] T. P. Cao, G. Deng, and D. Mulligan, "Implementation of real-time pedestrian detection on FPGA," in *23rd International Conference on Image and Vision Computing New Zealand(IVCNZ'08)*, Nov 2008, pp. 1–6.

[39] C. Wojek, G. Dorkó, A. Schulz, and B. Schiele, "Sliding-windows for rapid object class localization: A parallel technique," in *Proc. of the 30th DAGM symposium on Pattern Recognition*, Jun 2008, pp. 71–81.

[40] V. Prisacariu and I. Reid, "fastHOG - a real-time gpu implementation of HOG," Department of Engineering Science, Oxford University, Tech. Rep. 2310/09, 2009.

[41] F. Hosseini and A. Fijany, "Fast parallel implementation of hog-based human detection on a highly parallel, low power, SIMD architecture," Submitted to 25th IEEE International Parallel & Distributed Processing Symposium(IPDPS '11).

[42] N. Ntene and J. H. van Vuuren, "A survey and comparison of guillotine heuristics for the 2d oriented offline strip packing problem," *Discrete Optimization*, pp. 174–188, 2009.

[43] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2003.

BIOGRAPHY

Amir Fijany is a Senior Researcher in the TeleRobotics and Applications (TERA) Department of the Italian Institute of Technology (IIT). He joined IIT in June 2008. Prior to joining IIT, he was an Associated Tech Fellow at Northrop Grumman Corp. (January 2008-June 2008) and a Principal Research Scientist in and Technical Group supervisor of the Advanced Computing Algorithms and ISHM Technologies Group at the Jet Propulsion Laboratory, California Institute of Technology (July 1987-January 2008). He received his BS and MS degrees in Electrical Engineering from School of Engineering, Tehran University, and DEA and PhD degrees in Computer engineering from University of Paris XI (Orsay). His research interests include model-based diagnosis, sensor placement, new computing devices and paradigms, advanced algorithms for highly parallel computation, and quantum information processing. He is the author/coauthor of over 80 scientific and technical papers, and research monographs. He holds 11 patents (three pending) on model-based diagnosis, new computing devices, and advanced architectures for signal processing and robotics applications. He has received over 30 NASA Technical Innovation Awards.

Fouzhan Hosseini is a PhD candidate in Tele Robotics and Applications (TERA) at Italian Institute of Technology (IIT). She joined IIT in February 2009. She is currently working on the programming models and parallel algorithms suitable for emerging low-power highly parallel computer architectures to provide supercomputing level capability for robotic systems and other embedded image processing applications. She received the B.S. and M.S. degrees in computer engineering from School of Electrical and Computer Engineering, University of Tehran, in 2006 and 2008, respectively. Her research activities include parallel algorithms, parallel programming models, computer architectures, and autonomous and embedded systems. She was ranked as second best student among computer engineering students of University of Tehran, in 2006. She is the recipient of Tehran University award for the best student in 2005.