SD-VBS: The San Diego Vision Benchmark Suite

Sravanthi Kota Venkata, Ikkjin Ahn*, Donghwan Jeon, Anshuman Gupta,
Christopher Louie, Saturnino Garcia, Serge Belongie, and Michael Bedford Taylor
http://parallel.ucsd.edu/vision
Department of Computer Science and Engineering
University of California, San Diego

***Abstract*—In the era of multi-core, computer vision has emerged as an exciting application area which promises to continue to drive the demand for both more powerful and more energy efficient processors. Although there is still a long way to go, vision has matured significantly over the last few decades, and the list of applications that are useful to end users continues to grow. The parallelism inherent in vision applications makes them a promising workload for multi-core and many-core processors.**

While the vision community has focused many years on improving the accuracy of vision algorithms, a major barrier to the study of their computational properties has been the lack of a benchmark suite that simultaneously spans a wide portion of the vision space and is accessible in a portable form that the architecture community can easily use.

We present the San Diego Vision Benchmark Suite (SD-VBS), a suite of diverse vision applications drawn from the vision domain. The applications are drawn from the current state-of-the-art in computer vision, in consultation with vision researchers. Each benchmark is provided in both MATLAB and C form. MATLAB is the preferred language of vision researchers, while C makes it easier to map the applications to research platforms. The C code minimizes pointer usage and employs clean constructs to make them easier for parallelization.

Furthermore, we provide a spectrum of input sets that enable researchers to control simulation time, and to understand properties as inputs increase to leverage better processor performance.

In this paper, we describe the benchmarks, show how their runtime is attributed to their constituent kernels, overview some of their computational properties – including parallelism – and show how they are affected by growing inputs. The benchmark suite will be made available on the Internet, and updated as new applications emerge.

I. Introduction

Computer vision is an exciting application domain that has undergone extensive research and progress over the last few decades, and continues to evolve. Vision has found uses across a diverse and rich set of fields including medicine, automotive robotics, web search, guidance systems, and even care for the elderly. Some areas have been developed so much that they are considered "solved problems" and algorithms researchers have moved on to new areas.

In order to focus on the core algorithms, many vision researchers have long ago abandoned the requirement for real-time processing, let alone super real-time processing. At the same time, robotics researchers have long been accustomed to the realization that many of these algorithms will be out of reach for untethered stand-alone systems because of the impracticality of running these algorithms on the available computational platforms in real-time.

Recently, motivated by the power crisis brought on by transistor scaling limitations, the processor industry has adopted the multi-core and many-core processor paradigm as the chosen way of improving the performance of consumer computing systems. However, this performance can only be realized on those application domains that have sufficient levels of exploitable parallelism. It is these applications that will exploit the benefits of highly multi-core processors, and simultaneously, drive the demand for the next generation of Moore's Law.

Computer vision is a tantalizing application domain for multi-core processors because it offers the prospect of ample parallelism. Multi- and many- core processors, as well as specialized hardware solutions, provide the promise of bringing state-of-the-art vision algorithms to real-time, and even super-real time performance levels[1]. These kinds of computations are highly constrained by computation, cost and power on current systems. Enabling real-time vision in machines relies not only on the advancement of computer vision research but also on the improvement of the hardware and software platforms that will make vision viable in tethered and untethered systems within the system's energy and cost budget.

Even as a given vision application is made viable by improvements to our computational platforms, new directions exist which require continued increases in performance. More ambitious goals drive us to try more precise analyses, larger image sizes, more extensive databases of features to match against, and ultimately, super-real-time analysis of pre-existing footage. As a result, vision processing is an area that will continue to spur the growth of successively more advanced vision platforms.

In addition to many- core and multi- core processor platforms, specialized vision architectures have been an exciting area of research and commercial activity for several decades. In that time, vision platforms have evolved from MIMD computer boards [1] to complete SoC (System on Chip) implementations [2] and distributed system [3] that promise support for a variety of vision sub-areas. Commercial chips for computer vision have also been proposed and designed

* Now at Google Corporation.

[1]For instance, as Internet search companies would like to do with images on the Internet. In the same way that we want to process text at super-real time, so too does it makes sense to process images in super-real time.

commercially in the past, such as Sarnoff's Acadia [4] and MobilEye's EyeQ [2].

To foster understanding of vision workloads, whether for multi- or many- core systems, or for specialized platforms, or for compilation and run-time systems, we have developed a benchmark suite, called SD-VBS, which includes nine important vision applications drawn from a diverse set of vision domains. The nine applications themselves are in turn composed a collection of over 28 non-trivial computationally intensive kernels. The applications and kernels are shown in Figure I.

For each of these applications, we provide both MATLAB and C versions. The MATLAB is typically the original source language of the benchmarks, as coded by vision researchers. The C versions are provided to make it easier for architecture researchers to work with the benchmarks. They have been coded in a "clean" subset of C which avoids unnecessary use of pointers, legacy or machine-specific optimization, and C language features that make analysis and transformation difficult. The goal is to facilitate the analysis, transformation, parallelization, and simulation of the benchmarks by compiler and architecture researchers.

Each benchmark is provided with inputs of three different sizes, which enable architects to control simulation time, as well as to understand how the application scales with more difficult inputs. We have also provided several distinct inputs for each of the sizes, which can facilitate power and sensitivity studies. Section III examines execution properties of the benchmarks with the different inputs, in order to give intuition on benchmark kernel importance and scaling as inputs are changed and increased in size.

The specific contributions of this work are as follows:

- We introduce a new vision benchmark suite, which covers a rich set of applications and utilizes a comprehensive set of common vision kernels.
- We describe the benchmarks, their purpose and their basic structure.
- The benchmark is written in both MATLAB and "clean" C, the languages of choice for vision researchers and computer architects.
- High-level execution properties of each vision application are analyzed, providing insight into the workings of the applications and their kernels.

The rest of the paper is organized as follows. Section II describes the applications constituting the benchmarks and the vision kernels covered. Section III examines the characteristics of the applications. Section IV presents the related work, and the paper concludes with Section VI.

II. Vision Benchmark Suite

In our effort to build a comprehensive benchmark for computer vision domain, we consulted widely with vision and machine learning researchers and assembled a diverse set of vision applications. The San Diego Vision Benchmark Suite (SD-VBS) comprises nine applications employing over 28 kernels, with three different sets of configurations per application and five distinct input data sets per configuration. Moreover, both MATLAB and C versions for the SD-VBS suite have been provided. SD-VBS is available at http://parallel.ucsd.edu/vision.

TABLE I
Benchmark classification based on concentration area

Benchmark	Concentration Area
Disparity Map	Motion, Tracking and Stereo Vision
Feature Tracking	Motion, Tracking and Stereo Vision
Image Segmentation	Image Analysis
SIFT	Image Analysis
Robot Localization	Image Understanding
SVM	Image Understanding
Face Detection	Image Understanding
Image Stitch	Image Processing and Formation
Texture Synthesis	Image Processing and Formation

A. Design Philosophy

Applications within the SD-VBS suite were selected across the broader spectrum of vision algorithms, and strive to include the most frequently used kernels in computer vision area. These applications range from traditional algorithms to more up-to-date algorithms, and from stand-alone applications to base modules for bigger applications. We have chosen representative applications from each of the following vision concentration areas: *Image Processing and Formation*; *Image Analysis*; *Image Understanding*; and *Motion, Tracking and Stereo Vision*. This suite comprises code for vision modules, including Disparity Map, Feature Tracking, Image Segmentation, Scale Invariant Feature Transform (SIFT), Support Vector Machines (SVM), Robot Localization, Face Detection, Image Stitch, and Texture Synthesis. The kernels range from the basic operations such as convolution filters and area sum to the more complex Adaboost [5].

Table I summarizes the classification of each of the benchmark under the concentration area they represent. Every concentration area has two or three benchmarks associated with it and each benchmark is chosen such that it is one of the key representatives of the area. The breadth of the SD-VBS suite is apparent from the concentration areas it targets, while depth is created by the relative self-sufficiency of the individual benchmarks.

B. Applications

Currently SD-VBS comprises nine applications: Disparity Map, Feature Tracking, Image Segmentation, Scale Invariant Feature Transform (SIFT), Support Vector Machines (SVM), Robot Localization, Face Recognition, Image Stitch, and Texture Synthesis.

Table II gives a brief description of each benchmark and the application they target.

- **Disparity Map** [6]: Given a pair of stereo images for a scene, taken from slightly different positions, the disparity map algorithm computes the depth information

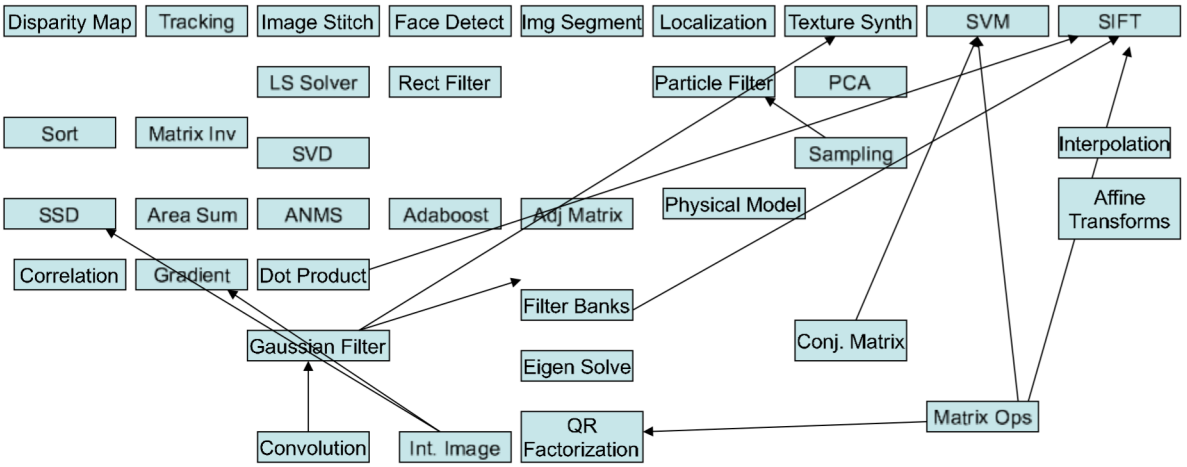

Fig. 1. **Decomposition of the Vision Benchmarks into their major kernels.** Each benchmark is listed at the top of the figure, and each benchmark's major kernels are listed directly underneath. An arrow is used to indicate cases where kernels are shared between applications.

TABLE II
BRIEF DESCRIPTION OF SD-VBS BENCHMARKS

Benchmark	Description	Characteristic[a]	Application Domain
Disparity Map	Compute depth information using dense stereo	Data intensive	Robot vision for Adaptive Cruise Control, Stereo Vision
Feature Tracking	Extract motion from a sequence of images	Data intensive	Robot vision for Tracking
Image Segmentation	Dividing an image into conceptual regions	Computationally intensive	Medical imaging, computational photography
SIFT	Extract invariant features from distorted images	Computationally intensive	Object recognition
Robot Localization	Detect location based on environment	Computationally intensive	Robotics
SVM	Supervised learning method for classification	Computationally intensive	Machine learning
Face Detection	Identify Faces in an Image	Computationally intensive	Video Surveillance, Image Database Management
Image Stitch	Stitch overlapping images using feature based alignment and matching	Data and computationally intensive	Computational photography
Texture Synthesis	Construct a large digital image from a smaller portion by utilizing features of its structural content	Computationally intensive	Computational photography and movie making

[a] We employ the term "data intensive" to characterize codes with repetitive low-intensive arithmetic operations across a very fine level data granularity. We employ the term "computationally intensive" to refer to those codes that are less predictable and perform more complex mathematical operations on a potentially more unstructured data set.

for objects jointly represented in the two pictures. The depth information in a scene gives clarity about relative positions of objects in the scene. Robot vision systems use Disparity Map extensively to compute the depth information, which is useful in applications such as cruise control [7], pedestrian tracking, and collision control.

The implementation is based on Stereopsis [6], also known as Depth Perception. Given a stereo image pair, the disparity algorithm computes dense disparity. Dense disparity operates on every pixel of the image (fine level data granularity) unlike sparse disparity where depth information is computed on features of interest. Figure I shows the constituent kernels of the disparity module: filtering, correlation, calculation of sum of squared differences (SSD) and sorting. For better cache locality, the 2-D filtering operation was implemented as two 1-D filters. Correlation and SSD are computed on every pixel across the image, making them expensive data intensive operations.

The code size is small compared to other benchmarks in SD-VBS, but the number of operations on fine grained pixel data is enormous for dense disparity. From a pro-

grammer's point of view, disparity has program characteristics such as regular and prefetch-friendly memory accesses and predictable working set. In conclusion, disparity is a parallelization-friendly algorithm whose performance is only limited by the ability to pull the data into the chip.

- **Feature Tracking** [8]: Tracking is the process of extracting motion information from a sequence of images. This involves feature extraction [9] and a linear solver that calculates the movement of features. This application is widely used in robotic vision and automotive domain for real-time vehicle or feature tracking.
  SD-VBS implements the Kanade Lucas Tomasi (KLT) tracking algorithm [8] for feature tracking. The algorithm comprises of three major computation phases: image processing, feature extraction and feature tracking. The image processing phase involves noise filtering, gradient image and image pyramid computations, which operate on pixel level granularity. The core of the algorithm – feature extraction and tracking – operates on coarse grained data, which is identified by the features.
  The working set varies significantly across the major computation blocks. While image processing operates on the entire image, the operations are restricted to Multiply-and-Accumulate (MAC) making it a data intensive phase of the application, yet parallelization-friendly. The feature extraction and tracking kernels operate on feature level granularity and the code is complicated by complex matrix operations such as matrix inversion and motion vector estimation, making it computationally intensive and making exploitation of innate parallleism more challenging.
- **Image Segmentation** [10]: Image Segmentation refers to the process of partitioning a digital image into conceptual regions. Segmentation is typically used in finding objects of interest such as boundaries, object characteristics. Each segment that is described by the algorithm comprises a set of pixels that share certain desirable characteristics. Image segmentation finds applications in medical imaging, face and fingerprint recognition, machine vision, computational photography etc.
  The algorithm implemented in SD-VBS can be divided into three sub tasks: construction of a similarity matrix, computation of discrete regions, and normalized segmentation. The similarity matrix is computed by analyzing pixel-pixel pairs. Based on this, we compute regions or classes of an image. Repetitive operations on pixel granularity make this phase a good candidate for the exploitation of ILP on fine granularity and DLP (data level parallelism) across iterations.
  The segmentation kernel involves discretization, complex arithmetic and matrix computations, all of which operate on a pixel level granularity. Overall, this application involves complex operations across fine granularity, thus making it compute intensive application.
- **SIFT** [11]: The Scale Invariant Feature Transform (SIFT) algorithm is used to detect and describe robust and highly distinctive features in images. Extraction of robust image features that are invariant to scaling, rotation and noise finds wide applicability in domains such as object recognition, image stitching, 3D modeling, video tracking. A very interesting and desired feature of SIFT descriptors is its robustness to partial occlusions, which makes it a pervasive algorithm for navigation and match moving applications.
  SIFT implements David Lowe's Scale Invariant Feature Transform algorithm in the SD-VBS. This algorithm computes keypoints (or features) and their descriptors given a gray scale image. The key kernels of the algorithm comprise image processing, keypoint detection and feature descriptor computation. The preprocessing stage of SIFT involves filtering operations in addition to a data-compute intensive linear interpolation phase (upsampling to extract anti-alias image). The detection of keypoint phase involves creation and pruning of the Difference of Gaussian (DOG) Pyramids. Creation of the DOG is data intensive while feature extraction is computationally intensive. The descriptor computation kernel employs histogram binning to assign orientations to feature points. This phase operates on "feature-level" granularity and has heavy computations.
  The image processing and DOG creation phase are characterized by regular prefetch-friendly memory access pattern and predictable working set. Identification of keypoints and descriptor assignment is plagued by irregular memory pattern and intensive computations. An interesting point to note here is that the irregular memory access can be hidden by the intensive computations, suggesting that we could in fact extract fine-grained parallelism.
- **Robot Localization** [12]: Robot Localization estimates the position of a robot relative to its environment based on sensor data. It is one of the fundamental problems of mobile robotics and is the key component in autonomous robot systems.
  This problem can be divided into two sub-tasks: global position estimation and local position tracking. Global position estimation is the ability to determine the robot's position in an a priori or previously learned map, given no other information than that the robot is somewhere on the map. Once a robot has been localized in the map, local tracking is the problem of keeping track of that position over time. The SD-VBS suite implements the Monte Carlo Localization algorithm (MCL). The MCL algorithm is a particle filter combined with probabilistic models of robot perception and motion.
  The kernels used in the application are particle filters and physical modeling. These kernels utilize trigonometric functions and thus make heavy utilization of floating point engines. The kernels operate at the granularity of positions. The irregularity in data access patterns and intense computations make localization a compute-

intensive application.

- **SVM** [13]: Support Vector Machines are a supervised learning method used for data classification. Given a set of data points and training data, SVM optimally separates the data into two categories with maximal geometric margin. SVMs are closely related to neural networks and belong to a family of generalized linear classifiers. Applications in machine learning employ SVM data classification extensively.
  The SVM benchmark in SD-VBS uses the iterative interior point method to find the solution of the Karush Kuhn Tucker conditions of the primal and dual problems [14]. The algorithm can be broadly classified into two phases: training and classification. Both these phases involve heavy polynomial functions and matrix operations. The non-deterministic nature of access pattern makes it difficult to classify SVM as either fine grained or coarse grained parallelism.
- **Face Detection** [15]: The Face Detection algorithm determines the locations and sizes of human faces in digital images. This algorithm can be regarded as a specific case of object-class detection. Face detection is used extensively in biometrics, video surveillance, human computer interface and image database management.
  The SD-VBS suite employs the Viola Jones Face Detection algorithm. This algorithm can be divided into three components: extract faces, extract face sequence and stabilize face windows. Kernel "extract faces" does image preprocessing and also extracts features. The granularity of these operations is pixel-level and the computations performed are complex. After this phase, the granularity changes to feature level and at this point, the memory accesses become irregular and unpredictable.
  We classify face detection as compute intensive due to its "feature-based" approach.
- **Image Stitching** [16]: Image stitching or photo stitching is the process of combining multiple photographic images with overlapping fields of view to produce a segmented panorama or high-resolution image. It is also known as mosaicing. Image stitching has applications in computational photography and movie making.
  Extraction of overlapping regions between images forms the basis for image stitch. The implementation of stitch algorithm is classified into four broad categories: image calibration and registration, feature extraction, feature matching and image blending. The image calibration phase involves filtering operations to process the image for the sensitive feature extraction phase. As discussed earlier, calibration is data intensive and can be exploited by fine grained parallelism.
  The feature extraction kernel involves preprocessing and feature selection. The preprocessing phase of the feature extraction kernel employs gradient filters on pixel-level granularity. The feature selection (ANMS kernel) is coarse grained as it operates on features rather than pixels. The regularity in access patterns breaks here and the stitch application enters the realm of heavy computations on irregular data points.
  The feature matching stage of the algorithm identifies exact overlap between a pair of images (intensive search) based on features. SD-VBSsuite uses the RANSAC [17] algorithm for the purpose of image matching. This algorithm is iterative, heavily computational and accesses data points randomly. Image blending also counts among computationally intensive kernels owing to heavy transform and image alignment operations.
  The image stitch algorithm is a classic example of a benchmark that has potential for all three types of parallelism: Instruction, Data and Thread Level parallelism (ILP, DLP and TLP). The filtering kernels and image transform operations have a good degree of ILP, whereas the feature point based computations in extraction and blending kernels are TLP sections. The iterative nature of the algorithm adds data level parallelism. Thus, we will classify stitch as both data and compute intensive application.
- **Texture Synthesis** [18]: Texture synthesis is the process of constructing a large digital image from a smaller portion by utilizing some features of its structural content. It finds application in computational photography, digital image editing, 3D computer graphics and movie making. For the SD-VBS application, we are using the Portilla-Simoncelli implementation of texture analysis/synthesis [19]. The algorithm can be divided into three major parts: image calibration, texture analysis and texture synthesis.
  Image calibration has been discussed extensively in the above benchmarks. The other two kernels - analysis and synthesis, employ complex matrix, arithmetic, and trigonometric operations iteratively. These computations are performed on pixel granularity for the entire working set making heavy use of memory.
  The texture synthesis algorithm is an interesting example of TLP, where each thread exploits ILP due to complex operations hiding memory latency. We classify texture synthesis as compute-intensive application.

III. Evaluation

A. Methodology

In the following sections, we profile the SD-VBS benchmark codes in order to analyze the relative contributions of the kernels to their respective vision applications. In addition to identifying the hot-spots for each benchmark, we provide detailed program characteristics and their effect on access patterns, parallelism and program speed-up.

The data was gathered on a Linux system whose characteristics are specified in Table III.

For all benchmarks we have used multiple input data sizes. The data sizes used are SQCIF (128x96), QCIF (176x144) and CIF (352x288). QCIF is roughly 2x larger than SQCIF, and CIF is roughly 2x larger than QCIF.

TABLE III
CONFIGURATION OF PROFILING SYSTEM

Feature	Description
Operating System	Linux 2.6.18-8.1.10.el5
Processors	Intel Xeon 2.2 GHz
L1 cache	8 KB, 4-way set associative, write back
L2 cache	512 KB, 8-way set associative
Memory	6 GB, 6 banks, DDR 200MHz Synchronous
Front Side Bus	400MHz

B. Hot Spot Evaluation

We measured the performance of the nine SD-VBS benchmarks across different input sizes and image types. Our analysis for SD-VBS has been profiled across 65 variants of the SD-VBS test vectors. Figure 3 summarizes the profiling results for each of the benchmarks. The figures show the code coverage of kernels across different input image sizes. Table IV attempts to characterize the characteristics of the parallelism in each kernel. We classify kernels based on the type of parallelism they exhibit and then present the results of a dynamic critical path analysis, similar to that found in [20], that attempts to approximate the potential amount of intrinsic parallelism in the application. This parallelism figure corresponds roughly to the speedup possible on a dataflow machine with infinite hardware resources and free communication.

We loosely categorize parallelism into Instruction Level Parallelism (ILP), Data Level Parallelism (DLP) and Thread Level Parallelism (TLP). We define ILP as fine-grained parallelism that can be exploited across a basic block. DLP loops are frequently vector operations across a huge data set. DLP loops are typically characterized by simple and repetitive computations with predictable data access patterns. Inter-iteration dependence may or may not exist. TLP is defined as the parallelism that is achieved when two or more independent complex tasks could be scheduled simultaneously. TLP code is usually harder to extract parallelism from, because of irregular data access patterns and the issue of memory dependences. Inter-thread dependence may or may not exist. A DLP loop can be converted into TLP but vice versa is not true.

- **Disparity Map**: Figure 3 and Table IV summarize the characteristics of the benchmark. As explained earlier, disparity is data-intensive application that operates on pixel granularity. This observation is portrayed by the scaling of the execution time with input image size in Fig 2. Integral image is an interesting kernel that witnesses negative slope with increasing working set size. This can be attributed to the existence of high amount of thread level parallelism within the kernel. Regular memory access, predictable working set and iterative nature of the DLP computations make disparity an ideal candidate for parallelism. The hierarchical parallelism that exists among kernel – inter-kernel TLP, intra-kernel DLP and ILP within each strand (or iteration) – is responsible for the high levels of parallelism indicated by Table IV.
- **Feature Tracking**: Figure 3 shows that the majority of program occupancy is taken by Gaussian Filter, Gradient and Integral Image kernels (refer to Fig 3). Also, being a data intensive benchmark, the kernels scale with increase in working set size with exceptions in the Integral Image and Matrix Inversion kernels. The Integral image kernel profits from the high amount of TLP within the kernel and strong ILP presence in its individual strands. The Matrix inversion kernel comprises transpose and matrix multiply operations. The former has high DLP parallelism and the latter has high ILP. Together, these favor matrix inversion operation. This claim can be verified from the summary of Table IV.
  The preprocessing and feature extraction kernels comprise 55% of the execution time, whereas kernels such as feature tracking occupy less than 30% of the program time.
- **Image Segmentation**: For the segmentation benchmark, we generated profiling numbers based on input sizes and also number of segments for each input image size. For fixed number of segments, the relative performance of SQCIF, QCIF and CIF image sizes are similar.
  Image segmentation is a compute intensive benchmark with complex matrix operations across the working set. The majority of program time is occupied by the computation of similarity matrix kernel, which is a classic candidate for ILP. Lack of DLP scenario affects the overall parallelism of the benchmark. The compute intensive nature of the benchmark is well captured by Figure 3, where the occupancy of individual kernels remain constant across various image sizes.
  As we increase the number of segments per image size, the execution time varies linearly. Thus, we conclude that segmentation is constrained by the number of image segments and not by the image size.
- **SIFT**: This benchmark operates on feature granularity and uses complex computational resources. The preprocessing stage involving filter operations and convolutions exhibit ILP and DLP. Kernels such as Interpolation, SIFT (that includes corner detection, feature extraction) exhibit ILP due to intense computation and irregular memory patterns.
  The majority of run-time is occupied by interpolation and SIFT kernels. These together account to 65% of the execution. Integral image and pre-processing constitute 17%. As expalined in the earlier benchmarks, the integral image kernel is high in TLP content, thus scaling across input sizes does not scale the percentage occupancy of this kernel. This behavior of the benchmark is well captured by Figure 2.
  The computation intensive nature of SIFT is well captured by Figure 3, which shows that the percentage occupancy of kernels does not vary with the change in input size though the run-time of the benchmark scales with input size.
- **Robot Localization**: In this benchmark, the run time

depends on the data set that is provided to the program. Each data point refers to the location co-ordinates. And depending on the nature of the data point, a particular motion model (or function) is invoked. Each model is comprised of different set of functions and thus, the run-time of the algorithm cannot be classified based on the input size. This is reflected in Figure 3 and Figure 2, where the increase in input size does not scale the execution time accordingly.

The major hot spot for this benchmark is the weighted sample function, which contributes to 50% of the execution time. This is the Sampling kernel of Localization benchmark. The other 50% of the time is spent in Particle Filter kernel of the application. Both these kernels use complex mathematical operations such as trigonometric functions, square root thus making heavy utilization of floating point engines.

- **SVM**: This benchmark is a compute intensive application that performs various matrix operations that dominate most of the execution time. This application involves examining and learning from a training set. Functions such as finding the polynomial fit, matrix operations, account for more than 50% of the execution time. These functions belong to Matrix Ops kernel of SVM application. The learning and training functions take up more than 30% of the program time and these fall under the Learning kernel.

  Due to the irregularity and randomness of working set, we cannot classify the kernels of SVM as DLP. But the iterative nature of the algorithm and complex computations that can hide the memory latency make SVM a good candidate for ILP and TLP exploitation. For the working set size of 500x64x4 and two iterations, Table IV shows expected speed up numbers for each kernel. The training kernel, which classifies the data points into two groups, projects higher parallelism than the learning kernel. For greater number of iterations on learning method (thus improving the quality of SVM), the overall parallelism of the learning kernel would scale up accordingly due to the scope of TLP across iterations.
- **Image Stitch**: Image stitch is an interesting application that involves the iterative non-deterministic RANSAC algorithm. This algorithm performs feature based image mosaic and thus, the execution time varies with the input image size and also the quality of the image. For smaller images, the number of distinct features are fewer and this limits the quality of the synthesized image and also the run time is small.

  Figure 3 shows the percentage occupancy of kernels and Figure 2 portrays the scaling contour of the benchmark with input sizes. The image calibration and feature extraction kernels occupy more than 50% of the program time. These kernels involve filtering and matrix operations on pixel granularity. This feature makes them parallelism friendly, which is evident from the high speed up number from Table IV.

  Kernels such as feature matching and image blending do not show promising speed up numbers because they operate on coarse granularity and they are limited by irregular data patterns and low DLP, TLP content. Thus, the expected parallelism for these kernels is not as high as calibration and extraction.

  As mentioned earlier, the stitch application is sensitive to the quality of the image. For the purpose of Figure 3 ,we generated the SQCIF and QCIF images by down-sampling the CIF image using averaging filters. This operation compromised the sharpness of the down-sampled images and resulted in ”low-featured” images. The consequence of this well captured by Figure 3 where the kernels for feature extraction and blending (LS Solver and SVD) fail to occupy any time due to the lack of robust features.
- **Texture Synthesis**: In this benchmark, we classify our test images based on the texture. We divide them into two classes - stochastic and structural. The execution time for all the image types is almost similar due to the fixed number of iterations for synthesizing each image. To get better insight, we profiled the benchmark with varying iteration count and output image size. The execution time across each image set does not vary linearly with the input size. Major hot-spots include texture analysis, kurtosis and texture synthesis that account to more than 60% of the execution time. From Figure 3, these appear under the Sampling kernel. The Matrix operations kernel occupies 30% of the program execution time. These functions are intensive computationally and operate on features level granularity.

IV. Related Work

A variety of researchers in industry and academia have pulled together a number of vision-related code corpuses. Perhaps the most noted and widely used benchmark suite is Intel OpenCV [21]. OpenCV is a highly-tuned library of vision primitives that was originally targeted for Intel x86 specific libraries.

Although a benchmark suite could presumably be assembled around OpenCV, its main disadvantages come from the extensive degree of hand-tuning that has been applied to OpenCV, which reduces its clarity significantly and subsequently makes it extremely difficult to analyze, transform, and parallelize, for which SD-VBS was explicitly created. The use of OpenCV in the authors' early architecture research efforts was the primary motivation for creating SD-VBS.

While OpenCV library is likely the broadest example of a corpus of vision code, there are other existing benchmark suites which target related domains, such as MediaBench [22]. Mediabench focuses on compression, encryption, encoding and communication algorithms. The Spec2000 [23] benchmark suite includes *facerec*, a face recognition algorithm.

Several benchmark suites exist not to characterize performance aspects of vision systems but rather their accuracy,

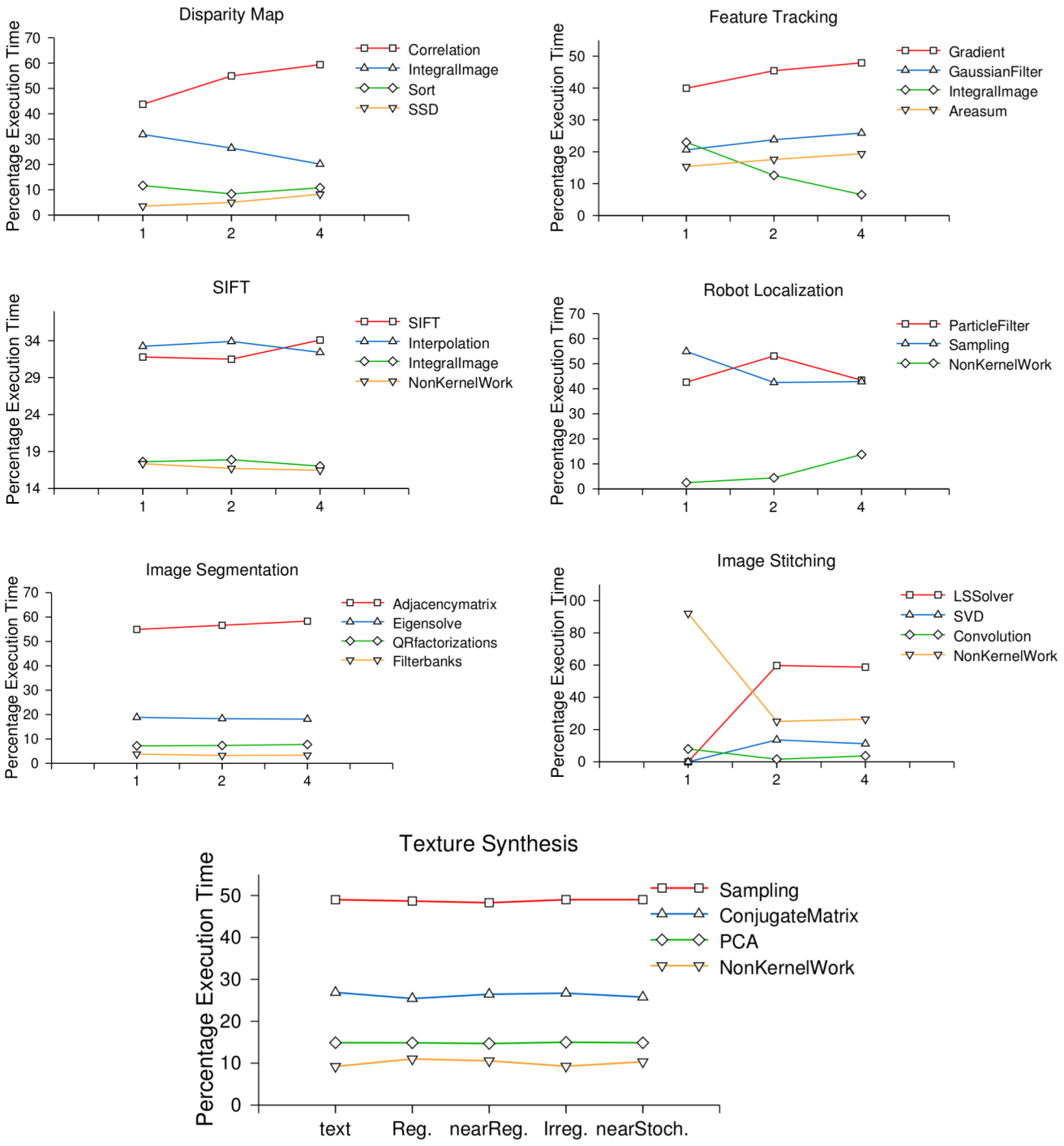

Fig. 3. **Benchmark hot spots.** The X-axis shows the relative input sizes. '1' refers to a SQCIF (128x96) image, '2' refers to a QCIF (176x144) image and '4' is a CIF image (352x288). The Y-axis represents the percentage occupancy of individual kernels. This figure captures the change in kernel occupancy as the input image sizes are scaled up. The nature of the variation slope characterizes the kernel property - data or compute intensive, along with the type of exploitable parallelism.

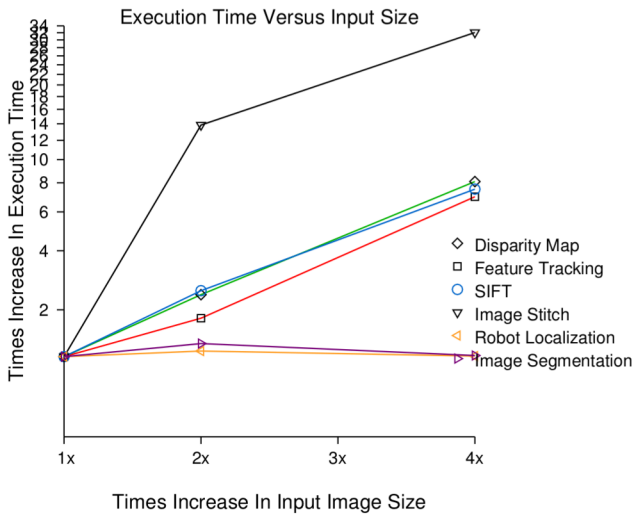

Fig. 2. Effect of data granularity on execution time. The X-axis shows the relative increase in the input size. The Y-axis shows the increase in the execution time. The graph captures the characteristic of the benchmark-programs that scale according to the input size are data-intensive and operate on fine granularity. Whereas those that are resistant to input size variation but are affected by the nature of the data are classified compute-intensive and must be parallelized with a coarser granularity.

Benchmark	Kernel	Parallelism Amount	Parallelism Type
Disparity	Correlation	502x	TLP
	Integral Image	160x	TLP
	Sort	1,700x	DLP
	SSD	1,800x	DLP
Tracking	Gradient	71x	ILP
	Gaussian Filter	637x	DLP
	Integral Image	1,050x	TLP
	Area Sum	425x	TLP
	Matrix Inversion	171,000x	DLP
SIFT	SIFT	180x	TLP
	Interpolation	502x	TLP
	Integral Image	16,000x	TLP
Stitch	LS Solver	20,900x	TLP
	SVD	12,300x	TLP
	Convolution	4,500x	DLP
SVM	Matrix Ops	1000x	DLP
	Learning	851x	ILP
	Conjugate Matrix	502x	TLP

TABLE IV
Parallelism across benchmarks and kernels THIS TABLE SHOWS THE PARALLELISM ESTIMATED BY A CRITICAL PATH ANALYSIS [20] IN THE VISION BENCHMARK SUITE. WHEN POSSIBLE, WE USED THE SMALLEST INPUT SIZE FOR THAT BENCHMARK, AND YET THERE ARE LARGE AMOUNTS OF INHERENT PARALLELISM.

including the Berkeley Segmentation Dataset and Benchmark [24], PEIPA [25], MUSCLE [26], and ImageCLEF [27].

V. SD-VBS USAGE

SD-VBS's balance between code cleanliness and performance is particularly useful in the multicore era, where a number of competing approaches have emerged for programming multicore processors, including transactions [28], shared memory, message passing, scalar operand networks [29], parallel languages [30], and others. For the purposes of comparison, the intent is that researchers should not change the algorithm but are permitted to restructure the code to suit their machines. When publishing data, users should also place on the Internet a diff of their benchmark code from the original.

VI. CONCLUSION

In this paper we have presented SD-VBS, a benchmark suite for computer vision, which is targeted toward researchers designing next-generation multi-, many- core and special-purpose systems, languages, and compilers.

The MATLAB and clean C implementations are intended to facilitate understanding and transformation of the code, both manual and automatically. We provide a spectrum of inputs to control simulation time, and to facilitate understanding the scaling of the algorithm. We hope that the benchmark suite will prove useful to compiler, architecture, and vision researchers alike, and help bring vision applications to "real-time" or to "super real-time."

Acknowledgments This research was supported in part by the National Science Foundation through Awards 0846152, 0811794, and 0725357, and by Calit2, the California Institute for Telecommunications and Information Technology. Implementation of Disparity Map [6], Feature Tracking [8], SVM [13], Robot Localization [12] and Image Stitch [16] were authored by Ikkjin Ahn. We would like to thank Andrea Vedaldi for the implementation of SIFT [11] and MSER; Athanasois Noulas for the implementation [31] of Viola Jones face detector [15]; Tom Mitchell for the data set for Viola Jones; Javier Portilla and Eero Simoncelli for their implementation of Texture Synthesis [19]; and Jianbo Shi for Image Segmentation implementation [32].

REFERENCES

[1] R. S. Wallace and M. D. Howard, "HBA Vision Architecture: Built and Benchmarked," in *IEEE Transactions on Pattern Analysis and Machine Learning*, March 1989.

[2] G. Stein et al., "A Computer Vision System on a Chip: a case study from the automotive domain," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2005.

[3] C. H. Chien and L. Lin, "PARADIGM: an architecture for distributed vision processing," in *International Conference on Pattern Recognition*, June 1990.

[4] G van Der Wal et al., "The Acadia Vision Processor," in *Proceedings of the International Workshop on Computer Architecture for Machine Perception*, September 2000.

[5] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," pp. 23–37, 1995. [Online]. Available: citeseer.ist.psu.edu/freund95decisiontheoretic.html

[6] D. Marr and T. Poggio, "Cooperative computation of stereo disparity," pp. 259–267, 1988.

[7] P. Chang, D. Hirvonen, T. Camus, and B. Southall, "Stereo-Based Object Detection, Classification, and Quantitative Evaluation with Automotive Applications," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2005.

[8] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of Imaging Understanding Workshop*, 1981.

[9] J. Shi and C. Tomasi, "Good features to track," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, Jun. 1994.

[10] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[11] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," in *International Journal for Computer Vision*, 2004.

[12] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA99)*, May 1999.

[13] J. Cui, Z. Li, J. Gao, R. Lv, and X. Xu, "The Application of Support Vector Machine in Pattern Recognition," in *IEEE Transactions on Control and Automation*, 2007.

[14] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, September 1999.

[15] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[16] R. Szeliski, "Image alignment and stitching: a tutorial," *Found. Trends. Comput. Graph. Vis.*, vol. 2, no. 1, pp. 1–104, 2006.

[17] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," in *Communications of the ACM*, June 1981.

[18] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *ICCV (2)*, 1999, pp. 1033–1038.

[19] J. Portilla and E. P. Simoncelli, "A parametric texture model based on joint statistics of complex wavelet coefficients," *Int. J. Comput. Vision*, vol. 40, no. 1, pp. 49–70, 2000.

[20] M. S. Lam and R. P. Wilson, "Limits of Control Flow On Parallelism," in *ISCA '92: Proceedings of the 19th Annual International Symposium on Computer Architecture*. ACM Press, 1992, pp. 46–57.

[21] G. Bradski, "The OpenCV Library," *Dr. Dobbs Journal of Software Tools*, 2000.

[22] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems," in *MICRO 30: Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 1997, pp. 330–335.

[23] SPEC, "SPEC CPU 2000 benchmark specifications," 2000, sPEC2000 Benchmark Release.

[24] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. 8th Int'l Conf. Computer Vision*, vol. 2, July 2001, pp. 416–423.

[25] "Pilot european image processing archive." [Online]. Available: http://peipa.essex.ac.uk/

[26] "Muscle benchmarking workpackage." [Online]. Available: http://muscle.prip.tuwien.ac.at/index.php

[27] M. Grubinger, P. D. Clough, H. Müller, and T. Deselaers, "The iapr benchmark: A new evaluation resource for visual information systems," in *International Conference on Language Resources and Evaluation*, Genoa, Italy, 24/05/2006 2006.

[28] M. Herlihy and J. E. B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, May 1993, pp. 289–300.

[29] M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalar Operand Networks," in *IEEE Transactions on Parallel and Distributed Systems*, February 2005.

[30] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: An efficient multithreaded runtime system," in *Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, Santa Barbara, California, Jul. 1995, pp. 207–216.

[31] A. K. Noulas and B. J. A. Krose, "E.M. detection of common origin of multi-modal cues," in *International Conference on Multimodal Interfaces*, 2006, pp. 201–208.

[32] "Matlab normalized cuts segmentation code." [Online]. Available: http://www.cis.upenn.edu/~ jshi/software/