CortexSuite: A Synthetic Brain Benchmark Suite*

Shelby Thomas, Chetan Gokhale, Enrico Tanuwidjaja, Tony Chong,
David Lau, Saturnino Garcia, and Michael Bedford Taylor
Department of Computer Science and Engineering
University of California, San Diego
Extended Version

Abstract— **These days, many traditional end-user applications are said to "run fast enough" on existing machines, so the search continues for novel applications that can leverage the new capabilities of our evolving hardware. Foremost of these potential applications are those that are clustered around information processing capabilities that humans have today but that computers do not yet have. The fact that brains can perform these computations serves as an existence proof that these applications are realizable. At the same time, we often discover that the human nervous system, with its 80 billion neurons, on some metrics, is more powerful and energy-efficient than today's machines. Both of these aspects make this class of applications a desirable target for an architectural benchmark suite, because there is evidence that these applications are both useful and computationally challenging.**

This paper details CortexSuite, a Synthetic Brain Benchmark Suite, which seeks to capture this workload. We classify and identify benchmarks within CortexSuite by analogy to the human neural processing function. We use the major lobes of the cerebral cortex as a model for the organization and classification of data processing algorithms. To be clear, our goal is not to emulate the brain at that level of the neuron, but rather to collect together synthetic, man-made algorithms that have similar function and have met with success in the real world.

To collect these benchmarks, we consulted six world-class machine learning and computer vision researchers, who collectively hold 83,091 citations across their distinct subareas, asking them to identify newly emerging computationally-intensive algorithms or applications that are going to have a large impact over the next ten years. We further found interesting real-world data sets that are representative of their use, and coded the benchmarks in "clean C" so as to make them accessible, analyzable, and usable for parallel and approximate compiler and architecture researchers alike. In this paper, we describe the benchmark suite–which extends the SD-VBS vision benchmark suite with eight more brain-centered applications–and provide an analysis of basic properties including the quantity of innate parallelism in each benchmark.

I. Introduction

With the increased availability of information, data science continues to focus on modeling complex relationships and extrapolating patterns. This has motivated research into a class of algorithms resembling the way our brain executes workloads in learning, pattern recognition, and sensory awareness. Although these tasks are trivial for the human brain to execute they traditionally result in prohibitively long execution times on modern processors. As multi-core and many core processors continue to improve, data-driven applications have come to the forefront of large-scale computing. There is a growing need for a comprehensive benchmark suite to help architecture researchers profile the behavior of these algorithms, realize how parallelization or energy-efficiency can be attained, and determine the scalability of these applications. [1]

In this paper, we present the Synthetic Brain Benchmark Suite (CortexSuite), which captures an emerging workload that is clustered around providing information processing capabilities that human brains have today but that computers are only just now beginning to become good at. We classify and identify benchmarks within CortexSuite by analogy to the human neural processing function, as shown in Fig 1. We use the major lobes of the cerebral cortex as a model for the organization and classification of data processing algorithms. *Our goal is not to emulate the brain at that level of the neuron, but rather to collect together man-made (i.e. synthetic) algorithms that have similar capabilities and have met with success in the real world.*

Benchmark Organization. In the organization of this benchmark suite we draw parallels between three major data processing classes and the four majors lobes of the sensory cortex: the temporal, occipital, parietal, and frontal. With this abstraction, we classify the occipital and temporal lobe, which work together to provide the visual processing centers of the brain, similar to a computer vision core. The parietal lobe, which is responsible for language comprehension, is parallel to a natural language core. Finally, we link the frontal lobe with learning and preprocessing benchmarks.

The benchmark suite contains eight unique applications from natural language processing, computer vision, and machine learning domains, and is intended to extend SD-VBS [1], which provides a core of relatively recent vision algorithm. The benchmarks are shown in Table I.

To account for the varied nature of data processing tasks, our proposed taxonomy helps to maintain consistency when extending this benchmark suite. For example, vision benchmark suites such as MEVBench [2] can be integrated into the occipital node of the suite and machine learning benchmark suites such as the ones in MineBench [3] and Sophia-ml [4] can be integrated into the frontal node.

Clean C coding. The applications have been coded to eliminate the use of unnecessary complex pointer operations

[1] *This paper extended from: Thomas, Shelby; Gohkale, Chetan; Tanuwidjaja, Enrico; Chong, Tony; Lau, David; Garcia, Saturnino; Bedford Taylor, Michael, "CortexSuite: A synthetic brain benchmark suite," Workload Characterization (IISWC), 2014 IEEE International Symposium on, pp.76,79, 26-28 Oct. 2014.

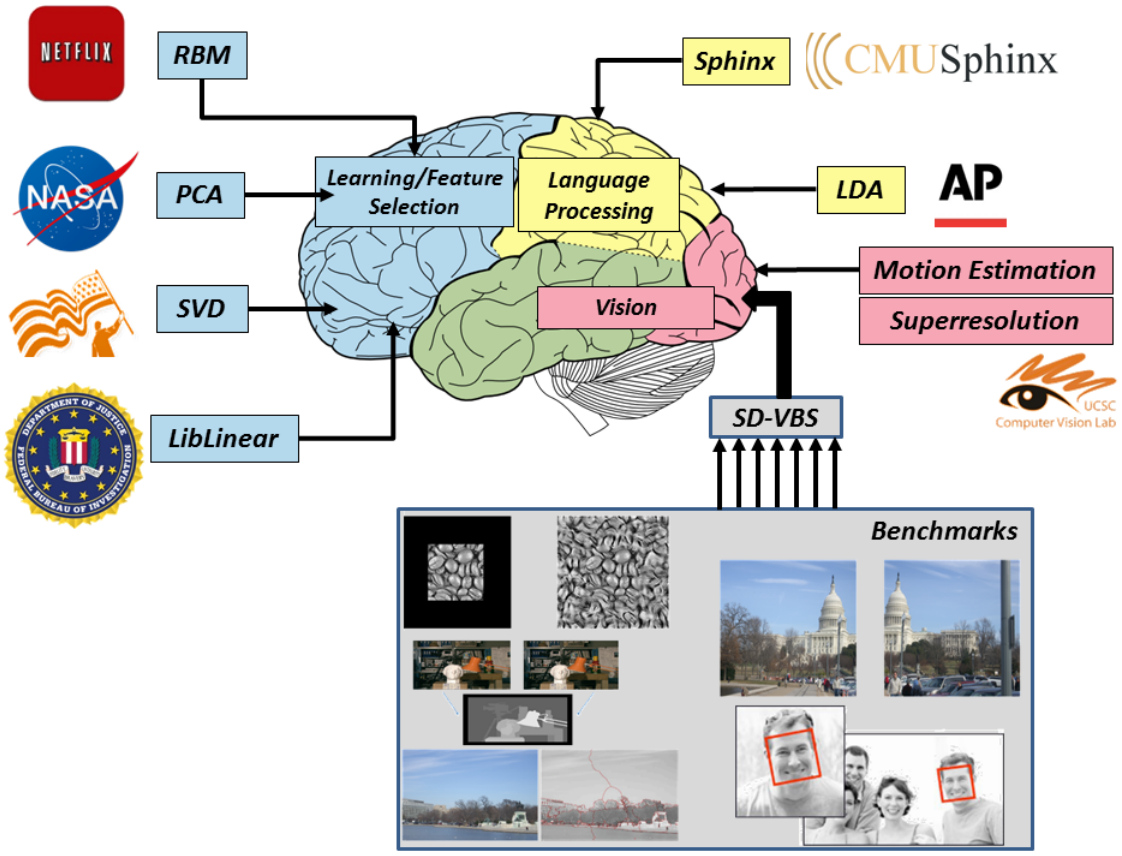

Fig. 1. ***The Synthetic Brain Benchmark Suite.*** CortexSuite captures an emerging workload that is clustered around providing information processing capabilities that human brains have today but that computers are only just now beginning to become good at. We classify and identify benchmarks within CortexSuite by analogy to the human neural processing function. We use the major lobes of the cerebral cortex as a model for the organization and classification of data processing algorithms. Our goal is not to emulate the brain at that level of the neuron, but rather to collect together man-made algorithms that have similar function and have met with success in real world use. To collect these benchmarks, we consulted six world-class vision and machine learning researchers, asking them to identify newly emerging computationally-intensive algorithms or applications that are going to have a large impact over the next ten years. This benchmark suite extends the SD-VBS vision benchmark suite [1] with eight more brain-centered applications–and provide an analysis of basic properties including the quantity of innate parallelism in each benchmark. In this diagram, we have mapped applications to the associated parts of the brain's cerebral cortex. Vision is processed primary by the occipital and temporal lobes, language processing by the parietal lobe, and learning/feature selection by the frontal lobe.

and machine-dependent optimization to expedite the use of the suite for parallel/approximate compiler and architecture researchers. Oftentimes, existing code bases have been tuned for a particular architecture, which obfuscates the underlying algorithm, making it difficult to retarget to a new execution model, and also preventing code transformations that can unlock parallelism.

Representative Inputs. Real-world datasets were identified to represent actual commercial and academic scenarios in an effort to mirror real-world use-cases. Moreover, we provided a spectrum of input sizes with varying run times and in some cases, different properties, for each benchmark, when it affects execution properties significantly.

- We develop a benchmarking framework for compute-intensive and useful emerging computations focused on brain-centric algorithms
- We create a benchmark suite that includes algorithms that are the current state-of-the-art
- We provide detailed explanations of each benchmark, its implantation and dataset
- We perform hotspot, parallelization and scalability analyses for each application

II. Benchmark Descriptions

The goal of the benchmark suite is to provide a set of algorithms that have the flexibility to cover a spectrum of domains with representative data in academic and commercial uses. In choosing the algorithms for the suite we consulted with researchers in the computer vision and machine learning fields from academia and industry. The benchmarks were coded to reduce the unnecessary use of pointer calculations, I/O and machine-specific optimizations that inhibit parallelization and/or their use in prototype compilers and/or architectural simulators. We omit the descriptions for the benchmarks that derive from SD-VBS [1] for the purposes of space.

CortexSuite contains eight different applications in addition to those in SD-VBS with three dataset sizes: small, medium,

TABLE I

Benchmarks in the Synthetic Brain Benchmark Suite. CORTEXSUITE CONTAINS NINETEEN BENCHMARKS THAT SEEK TO ENCOMPASS IMPORTANT NEW APPLICATIONS; IT INCLUDES EIGHT NEW BENCHMARKS (TOP) AND A NUMBER OF VISION-BASED BENCHMARKS FROM SD-VBS (BOTTOM).

Benchmark	Category	Dataset	Application Domain
Restricted Boltzmann Machines	Deep Learning	Netflix	Machine Learning
LibLinear	Classification/Regression	FBI Crime Statistics	Machine Learning
Principal Component Analysis	Feature Selection	NASA	Machine Learning
Singular Value Decomposition	Feature Selection	KOS Press	Machine Learning
Sphinx Speech Recognition	Speech Recognition	CMU	Natural Language Processing
Latent Dirichlet Allocation	Topic Modeling	Associated Press	Natural Language Processing
Super Resolution Reconstruction	Image Reconstruction	MDSP Research	Computer Vision
Motion Estimation	Motion, Tracking	MDSP Research	Computer Vision
Disparity Map	Motion, Tracking and Stereo Vision		Computer Vision
Feature Tracking	Motion, Tracking and Stereo Vision		Computer Vision
Image Segmentation	Image Analysis		Computer Vision
SIFT	Image Analysis		Computer Vision
Robot Localization	Image Understanding		Computer Vision
SVM	Image Understanding		Computer Vision
Face Detection	Image Understanding		Computer Vision
Image Sitch	Image Processing and Formation		Computer Vision
Texture Synthesis	Image Processing and Formation		Computer Vision

and large, corresponding roughly to 1, 10, and 100 seconds of execution time on a modern machine. We include multiple algorithms for applications in natural language processing, computer vision, and machine learning. In addition, a spectrum of unique and useful computer vision applications have also been provided through SD-VBS, summarized in Table I.

Restricted Boltzmann Machines (RBM) has seen exponentially growing use over the last couple of years in the context of deep learning networks. It is a stochastic neural network algorithm with applications in collaborative filtering [5], feature learning and topic modeling. The RBM algorithm was used in the Netflix prize's winning solution in 2009 [6], where Netflix provided their database of films, with user rankings, and offered $1 million for the best algorithm that would predict ratings of films that the user had not yet rate. RBM featured prominently in many of the top solutions that emerged over time.

CortexSuite utilizes RBM to implement movie suggestions (collaborative filtering) on variants of the Netflix database and provides the benchmark for the training process of RBM, the most computationally intensive aspect. To scale the run-time, we provide several scaled versions of the database; discarding minimally-connected films that have little impact on training. To train the RBM, we initialize the model with random parameters and repeat the training iteration until convergence. In every iteration, the current RBM tries to reconstruct the input data and then its parameters are adjusted based on the error of the data reconstruction. With more iterations, the reconstruction error diminishes, and we stop the training when the error converges or until a predetermined number of iterations is reached.

Sphinx Speech Recognition is used for the translation of spoken words to text [7]. This is achieved by taking the raw waveform, splitting it on utterances by silences, and attempting to recognize the word in each utterance [8]. We group all possible combinations of words and the best matching combination is chosen.

Super-resolution Reconstruction (SRR) is based on the idea that slight sub-pixel variations in the information encoded in a series of low resolution (LR) images can be used to recover one high resolution (HR) image [9], [10]. Computational resolution enhancement has many applications in the fields of photography, healthcare, security, astronomy, and military. Figure 4 shows an example of SRR in action.

Latent Dirichlet Allocation (LDA) is a topic modeling algorithm that is commonly found in natural language processing to discover topics from unordered documents. The underlying algorithm uses the assumption that each document was generated using a Dirichlet Distribution, which serves as a prior distribution to the Multinomial Distribution[12]. The goal of the algorithm is to find values for the multinomial parameter vectors and topics for each document.

Singular Value Decomposition (SVD) is a rank reduction algorithm used in many artificial intelligence, signal processing, and computer vision applications [13]. SVD is a factorization of a matrix into three matrices, as shown in Figure 2. The resulting three matrices have useful information

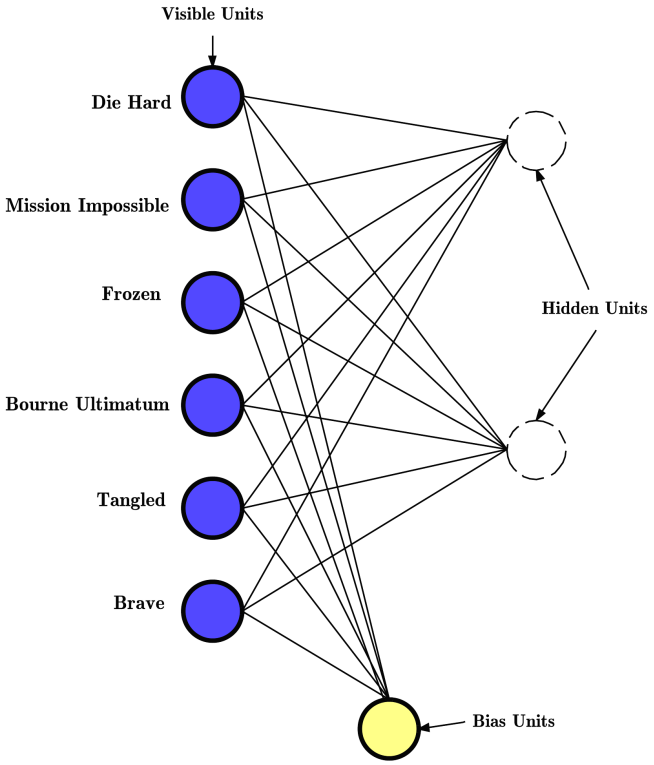

Fig. 2. **RBM Hidden and Visible Nodes:** Visible nodes represent movies while hidden nodes represent genres for each movie. Each movie will have an edge to each genre where the edges represents the weight of the connection between movie and genre. The value for each weight determines how correlated a movie is to a specific genre. Note: The output unit and other hidden units have been omitted from this picture.

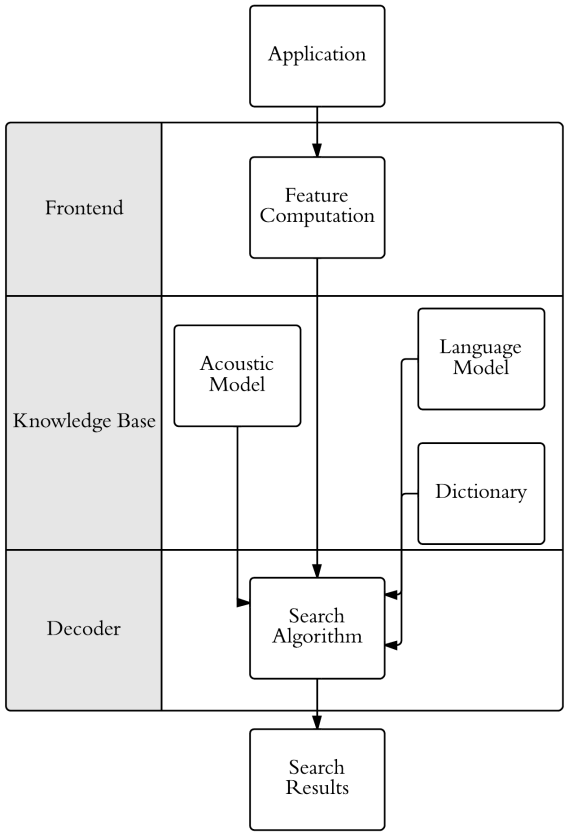

Fig. 3. **Sphinx Speech Recognition:**Sphinx first extracts features from speech waveform into feature vectors. The search algorithm will use the features to determine possible words using the acoustic model (HMM), and then combine the possible words into sentences using the language model (n-gram) and dictionary.

about the original matrix and can be multiplied together to obtain the original matrix back. We utilize SVD for Latent Semantic Analysis (LSA) [14]. LSA is a technique used in natural language processing to analyze relationships between a set of documents and the terms they contain.

Principle Component Analysis (PCA) is one of the most versatile and widely used statistical techniques for feature extraction in multivariate datasets. When dealing with high-dimensional data, one needs to consider the weight and importance of variables for accuracy and run-time considerations. PCA allows high-dimensional data to be reduced to an orthogonal lower-dimensional data-set by leveraging the dependencies between variables and identifying the most important ones, as shown in Figure **??**. As a prepossessing technique, PCA has found applications ranging from computer vision to machine learning and its tendency to use several matrix operations cause bottlenecks but in turn make it easy to parallelize.

Motion Estimation is the process of finding motion vectors that describe the transformation of one 2D image to another. It is an inverse, ill-posed problem as the motion is in 3D but the images are the projection of the 3D scene onto a 2D plane. Motion Estimation is an essential element in image and video processing. It is a key part of the video coding and applications such as frame rate up conversion, image super resolution etc. Performance of motion estimation can directly affect the performance of these applications.

Liblinear is a versatile library for large-scale linear classification and regression with applications in computer vision, natural language processing, neuroimaging, and bioinformatics. The library has been used in a variety of applications from real-time object recognition to predicting protein solubility and supports linear SVM, linear SVR and logistic regression. Liblinear is especially powerful for large scale data, i.e. with a large number of instances and features, as it is much faster (100$\times$ faster than libSVM) than other state-of-art linear or nonlinear SVM libraries [15] while keeping high accuracy.

In our variation of Liblinear we employ datasets to run binary and multi-class classification as well as regression. Binary linear classification is a supervised learning model in which elements are classified into two groups of data based on some predetermined metric. The goal of binary classification

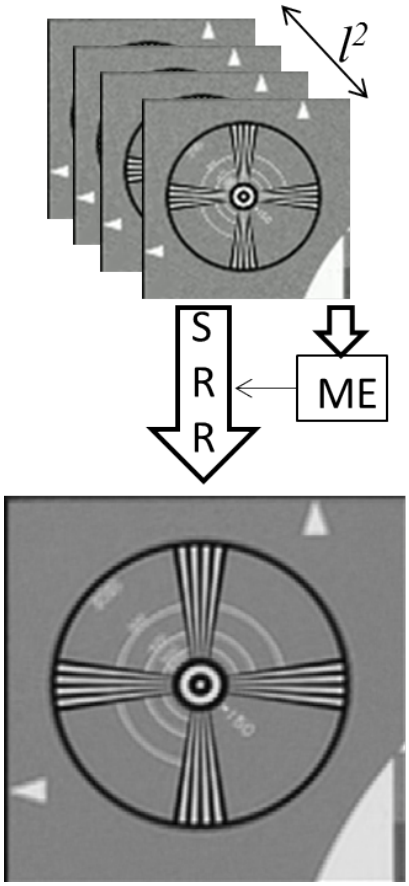

Fig. 4. ***Super-Resolution:*** 16 distinct LR images of size (128x96) are used to generate one HR image of size (512x384). Dataset courtesy: Multi-Dimensional Signal Processing (MDSP) research lab at the University of California, Santa Cruz [11]

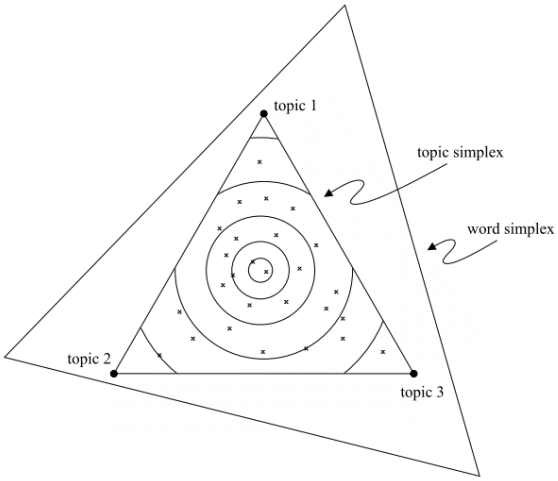

Fig. 5. ***Latent Dirichlet Allocation[12, pg. 1002]:*** Topic simplex for three topics inside of a word simplex for three words. Each corner in the topic simplex represents a specific topic and each point represents a single document. Each point inside of the topic simplex denotes a probability towards each topic, where each corner represents a probability of one for a certain topic.

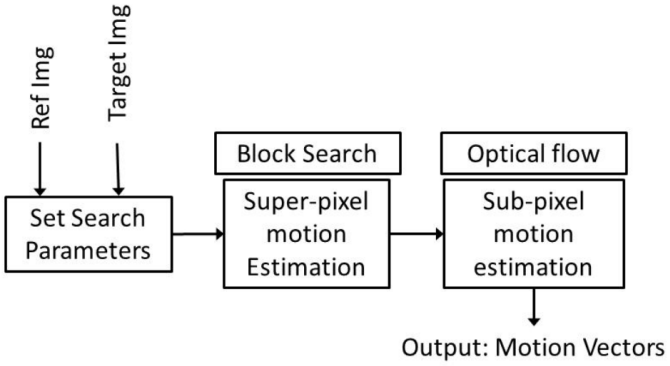

Fig. 6. **Motion Estimation:** Two images [ref, target] are input and output is a set of motion vectors. Full-pixel motion is estimated using block search algorithm. Sub-pixel motion is estimated using optical flow.

is to take a set of n-features and create a binary response which can then be used to reason about future sets of related data. Liblinear also supports multiclass classification through the one-vs-all which is a subset of binary classification. In a one-vs-all algorithm, each class is separated from others in training step. In the prediction step a simple binary classifier is run over each class and the one with the highest probability is chosen. Figure **??** shows an example of LibLinear in action.

III. Scalability Analysis

A primary goal of the Synthetic Brain Benchmark Suite is to provide researchers with a platform for evaluating the scalability of various human-inspired data processing tasks. Toward that end we analyzed all applications in the benchmark suite to identify these regions of the program that act as performance bottlenecks as the dataset size increases. To gather this data, we used Intel's VTune to profile the execution of the small, medium, and large datasets on a computer with an Intel Core i7-2620M CPU and 8GB RAM running Linux 3.13 and gcc 4.8.2 with O3. With these results we identified which regions of the program dominate runtime as data scales exponentially, the so called kernels of the applications. Figures 8 and 7 illustrate the relative and total execution times obtained from VTune profiling.

After obtaining kernel execution times, we used Kremlin [16] to perform critical path analysis [17] with the goal of quantifying the total amount of parallelism inside of the kernels. A critical path analysis approximates the potential amount of intrinsic parallelism in an application using a dynamic data flow analysis and a shadow memory to attach "earliest computation times" to each dynamic operation in the program. This parallelism figure corresponds roughly to the speedup possible on a dataflow machine with infinite hardware resources and free communication. This total parallelism–summarized in Table II–provides an approximate upper-bound on how well the program might scale on multi- and many core processors, or custom hardware accelerators. We use the "small" input sets.

Of note is that the vision benchmarks tend to have significantly more parallelism than the non-vision apps (exception: PCA), even though the input side runs larger, which aligns with our intuition.

In the following sections we will discuss the profiling results for individual benchmarks.

TABLE II
Parallelism Analysis. A CRITICAL PATH ANALYSIS ON EACH PROGRAM KERNEL DETERMINES THE TOTAL AMOUNT OF PARALLELISM AVAILABLE IN THAT REGION. THE "TOTAL PARALLELISM" IS THE IDEAL SPEEDUP FROM PARALLELIZING THAT REGION.

Algorithm	Functions	Total Parallelism
RBM	Gradient Descent	81
	Activate Hidden Node	128
	Activate Visible Node	39
SVD	QR Transform	327
PCA	Matrix Reduce	203
	Triangular decomposition	16,217
	Correlation Matrix	478,904
LDA	Inference	182
	Likelihood	303
LibLinear	SVC Solve	132
ME	FullSearch	344
	TaylorApp	27
SRR	Gauss-Siedel	15
	MatMul	919
	Read LR Pixels	38
Sphinx	Prune Channel	63
	Viterbi	5
	HMM	6
Disparity	Correlation	520
	Integral Image	160
	Sort	1,700
	SSD	1,800
Tracking	Gradient	71
	Gaussian Filter	637
	Integral Image	1,050
	Area Sum	425
	Matrix Inversion	171,000
SIFT	SIFT	180
	Interpolation	502
	Integral Image	16,000
Stitch	LS Solve	20,900
	SVD	12,300
	Convolution	4,500
SVM	Matrix Ops	1000
	Learning	815
	Conjugate Matrix	502

A. Sphinx

Sphinx speech recognition has a number hidden markov model computations that cause slowdown in the application. The majority of the program is located in two main operations, the evaluation of finding the optimal HMM sequence using the Viterbi algorithm, a dynamic programming algorithm [18] and the search of most likely sentence using n-gram model [19].

As the sentence length increases the bi-graph search and channel pruning go up considerably as both of these are functions of the data size. As the sentence length increases, the time it takes to perform the n-gram searches understandably starts to rise and the size of the hidden markov model grows. In Sphinx we find that the only real opportunity for parallelization lies in the channel pruning algorithm.

B. Restricted Boltzmann Machines (RBM)

Due to the nature of the RBM algorithm (generative stochastic neural network) most of the execution time is spent in various activation functions for hidden and visible units, taking up 40% of the total execution time. In a neural network, each processing unit is a neuron which fires a binary signal when it has reached certain parameters. Based on the type of activation function in the network, a function is specified which cause the neuron to fire a '1' or a '0' dependent on the function threshold. When this happens these signals propagate through the network and can cause other neurons to fire as well. These activation functions are integral in the training of neural networks. The gradient descent algorithm (SGD) is employed in the training of RBM and takes about as much time as the activation nodes and scale up the same way.

Parallelism analysis results (Table II indicate there is significant parallelism throughout the RBM kernels. For this analysis we split the activation functions into activate hidden and activate visible and found total parallelism values of 128 and 39, respectively. SGD's total parallelism was 81.

C. Latent Dirichlet Allocation (LDA)

LDA [20] suffers from being relatively floating point divide active with 14% of the execution time going into these long latency mathematical operations. The digamma function–the logarithmic derivative of the gamma function–also accounts for 30% of the execution time, as shown in Figure 8.

One unique aspect of LDA is that its non-kernel functions dominate execution time. This may be due to the fact that LDA has a large memory footprint and as the dataset size increases, the amount of memory being consumed increases as well, causing this memory function to increase significantly. We can conclude that the version of LDA that we have implemented may not provide the best scalability, as it is not practical to parallelize non-kernel functions.

D. Motion Estimation (ME)

We identify two primary kernels in ME: FullSearch and TaylorSeries. FullSearch is a block-matching kernel for full-pixel motion estimation while TaylorSeries is the core of optical flow to find out sub-pixel motion. Both of these kernels can be parallelized, with the latter exhibiting an order of magnitude

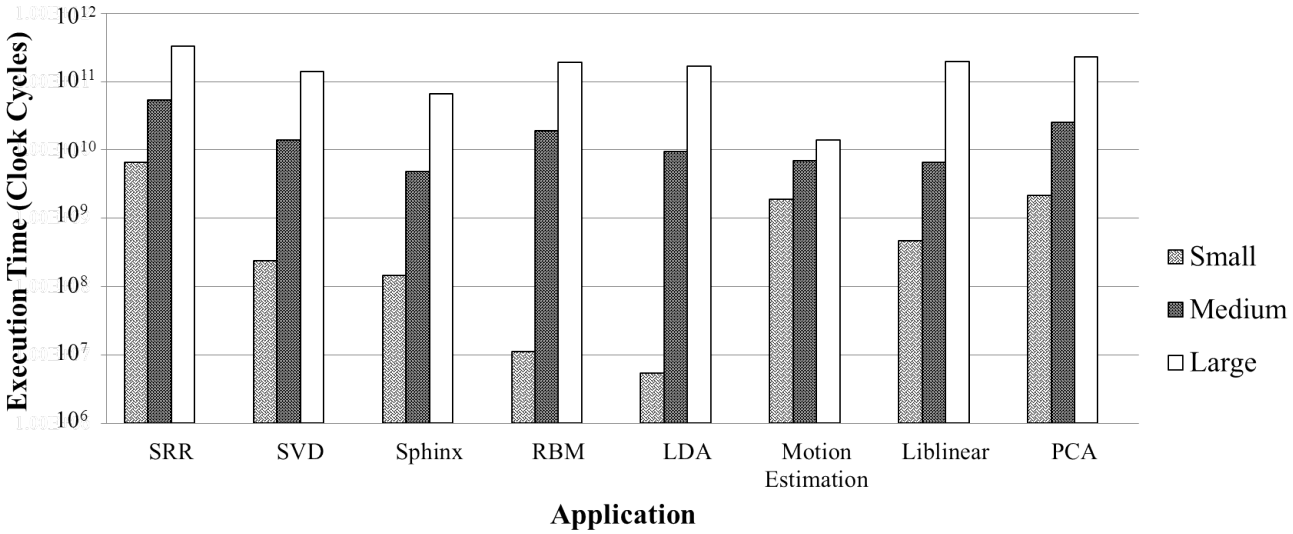

Fig. 7. ***Total Execution Times.*** To ensure kernel functions are sufficiently saturated, the large dataset for each algorithm was crafted to take a significant amount of time. This enabled us to see a trend in relative kernel execution time with increasing data. The total number of cycles is measured on an Intel Core i7-2620M CPU running at 2.70GHz.

more total parallelism. Examining this benchmark we found both coarse-grained (multiple blocks searched in parallel) and fine-grained parallelism (e.g. the matching operation of each block can be parallelized).

E. Sphinx

Sphinx speech recognition has a number hidden markov model computations that cause slowdown in the application. The HMM in this program take close to 30% of the total runtime, the majority of which is located in two main operations. This includes the evaluation of finding the optimal HMM sequence using the Viterbi algorithm, a dynamic programming algorithm [18] and the search of most likely sentence using n-gram model [19].

As the sentence length increases the bi-graph search and channel pruning go up considerably as both of these are functions of the data size. In addition, the time it takes to perform the n-gram searches understandably starts to rise and the size of the hidden markov model grows. In Sphinx we find that the only real opportunity for parallelization lies in the channel pruning algorithm. Virterbi is well known to lack parallelism.

F. Principle Component Analysis (PCA)

PCA's bottleneck comes from the fact that it must perform a myriad of matrix operations to obtain the principle components before finding the eigenvectors. Both of these steps, have very poor scalability as the number of operations that need to execute is exponential relative to data.

Due to the nested for loops in the matrix operations, we find that PCA is highly parallel. The creation of just the correlation matrix has total parallelism of $\sim$478,900.

G. Single Value Decomposition (SVD)

SVD combines various linear algebra operations to perform rank reduction. SVD's runtime is heavily data dependent. In particular, most of the execution time is spent in QR decomposition used to solve the least squares problem [21].

Parallelism analysis indicates that parallelization of SVD provides significant opportunity. The large amount of parallelism (327) is a result of the iterative nature of QR factorization. As the dataset size increases, the portion of the application related to the QR decomposition goes up proportionally.

H. Super-Resolution Reconstruction (SRR)

About 30% of SRR's time is spent performing the Gauss-Seidel and 45% of the time is spent on matrix operations. Figure 8 indicates that as the dataset size increases in SRR, the number amount of time spent in each kernel stay fairly consistent. To evaluate parallelism, Table II shows that the matrix multiply kernel exhibits significant amounts of parallelism (919). It is also possible to solve all the odd and even low resolution pixels in parallel (total parallelism: 38) but the Gauss-Siedel solver is largely serial in nature.

I. LibLinear

We ran Liblinear with a support vector machine workload and found that nearly 90% of its execution time is in a solver function (SVC Solve) that is responsible for regularization and running a support vector clustering algorithm. The SVC algorithm is responsible for mapping data from a lower to a higher dimension in order to find a suitable hyperplane that can provide a linear classifier between datasets. L2 regularization

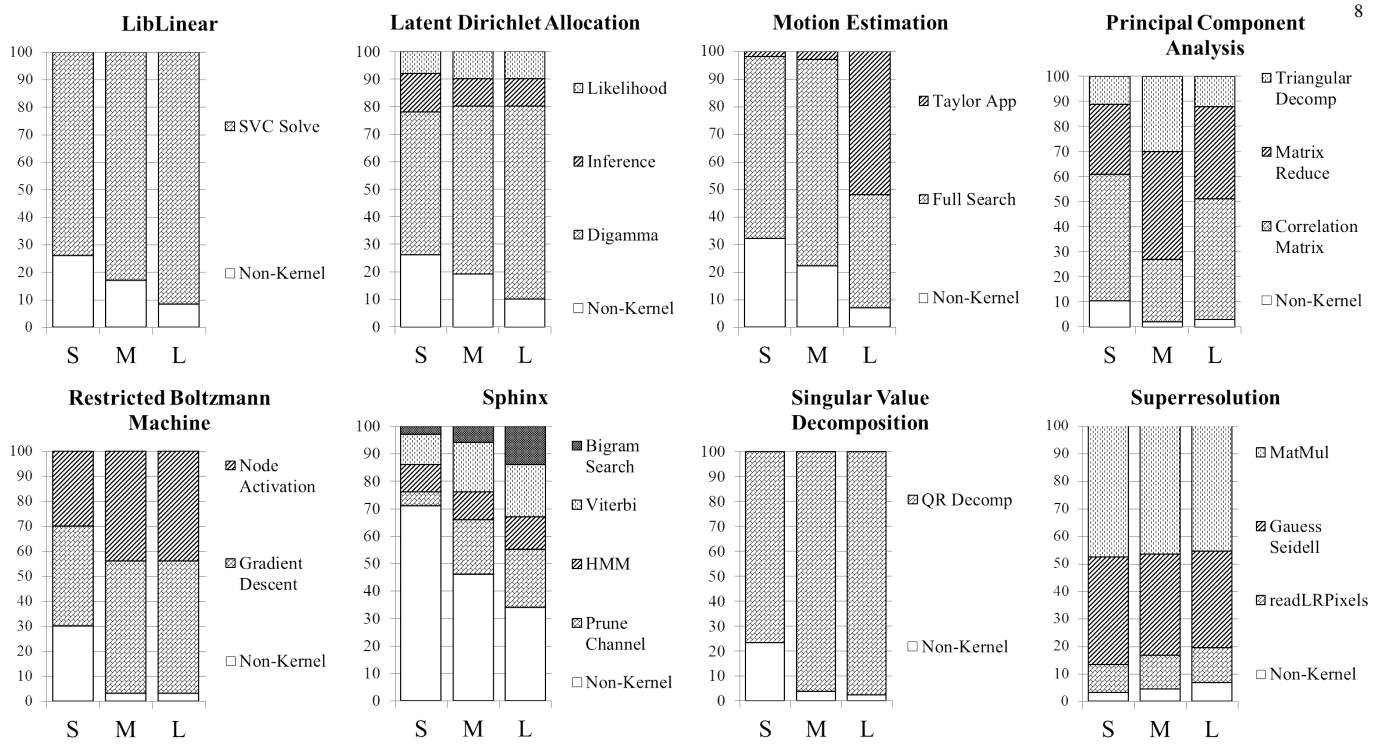

Fig. 8. ***Relative Execution Times.*** The graphs show the percentage of execution time spent in each of the major program kernels across the small (s), medium (m), and large (l) datasets. Time spent in the kernels dominates, especially in the large dataset where the average is 76.45%.

is an optimization technique used to prevent overfitting and reduce the complexity of the predictor.

As the inputs starts scaling up, it takes a fairly large amount of data for the solver function to saturate completely and when it does it dominates the program. Some of this slowdown can be resolved by exploiting the total parallelism of 132 in the solver function.

IV. Related Work

Many benchmark suites have been assembled in order to display the capabilities of machine learning algorithms which are tailored to specific tasks, such as computer vision. The San Diego Vision Benchmark Suite compiles a variety of computer vision algorithms, from SVM to Face Detection. The San Diego Vision Benchmark Suite gains an architectural understanding of a diverse set of computer vision algorithms and characterizes the computational properties of each algorithm. Reflecting its important a number of other computer vision related benchmarks including MEVBench [2] and MediaBench [22] have emerged. All of the aforementioned benchmark suites focus primarily on aspects of computer vision. Specifically, MEVBench focuses on mobile computer vision algorithms geared toward embedded systems and MediaBench focuses on video and multimedia processing.

Several other benchmark suites have focused on the architectural characterization of various machine learning algorithms. Most notably, the MineBench [3] benchmark suite has studied Data Mining and Bioinformatic workloads in order to characterize the branch and cache performance of each algorithm. Splash-2 [23] characterized parallel applications in terms of the cache performance on multiprocessors.

CortexSuite differs from existing suites in that it strives to complete the picture of a synthetic brain while also providing architectural analysis and an analysis of scalability of each algorithm. The datasets used in this benchmark suite provide real-world applications of each algorithm within the suite.

V. Conclusion and Future Work

CortexSuite captures an fascinating workload that is clustered around emerging algorithms that perform information process capabilities that traditionally have been relegated to human brains. We classify and identify benchmarks within CortexSuite by analogy to the human neural processing function. We use the major lobes of the cerebral cortex as a model for the organization and classification of data processing algorithms. Our goal is not to emulate the brain at the level of the neuron, but rather to collect together man-made algorithms that have similar function and have met with success in the real world. To collect these benchmarks, we consulted six world-class vision and machine learning researchers, asking them to identify newly emerging computationally-intensive algorithms or applications that are going to have a large impact over the next ten years. This benchmark suite extends the SD-VBS vision benchmark suite [1] with eight more brain-centered applications and provides an analysis of basic properties including the quantity of innate parallelism in each benchmark.

As the benchmark suite evolves we hope to help researchers get one step closer to realizing computers that outpace even

human capabilities in this new domain. Our open-source benchmark suite and the datasets will be released upon publication of this paper.

VI. Acknowledgments

This work was partially supported by NSF Awards 0846152, 1018850, and 1228992, and by C-FAR, part of STARnet, a Semiconductor Research Corporation program.

References

[1] S. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. Taylor, "SD-VBS: The San Diego Vision Benchmark Suite," in *IISWC*, Oct 2009.

[2] J. Clemons, H. Zhu, S. Savarese, and T. Austin, "Mevbench: A mobile computer vision benchmarking suite," in *IISWC*, Nov 2011, pp. 91–102.

[3] B. Ozisikyilmaz, R. Narayanan, J. Zambreno, G. Memik, and A. Choudhary, "An architectural characterization study of data mining and bioinformatics workloads," in *ISSWC*, Oct 2006, pp. 61–70.

[4] D. Sculley and G. Inc, "Large scale learning to rank," in *In NIPS 2009 Workshop on Advances in Ranking*, 2009.

[5] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: ACM, 2007, pp. 791–798.

[6] A. Tscher, M. Jahrer, and R. M. Bell, "The BigChaos Solution to the Netflix Grand Prize," 2009.

[7] X. Huang, F. Alleva, H.-W. Hon, M.-Y. Hwang, K.-F. Lee, and R. Rosenfeld, "The SPHINX-II speech recognition system: an overview," *Computer Speech & Language*, vol. 7, no. 2, pp. 137–148, 1993.

[8] Carnegie Mellon University, "Cmusphinx." [Online]. Available: http://cmusphinx.sourceforge.net/wiki/tutorialconcepts

[9] S. C. Park, M. K. Park, and M. G. Kang, "Super-resolution image reconstruction: a technical overview," *Signal Processing Magazine, IEEE*, vol. 20, no. 3, pp. 21–36, May 2003.

[10] Q. Zhang, R. Guy, and R. Plemmons, "Matrix structures and parallel algorithms for image superresolution reconstruction," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 4, pp. 1873–1893, 2010.

[11] P. Milanfar, "Mdsp super-resolution and demosaicing datasets." [Online]. Available: http://www.soe.ucsc.edu/~milanfar/software/sr-datasets.html

[12] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, January 2003.

[13] Strang, *Introduction to Linear Algebra*. Wellesley-Cambridge Pr., 2003.

[14] S. T. Dumais, "Latent semantic analysis," *Annual Review of Information Science and Technology*, vol. 38, no. 1, pp. 188–230, 2004.

[15] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A Library for Large Linear Classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, Jun. 2008.

[16] S. Garcia, D. Jeon, C. Louie, and M. Taylor, "The Kremlin Oracle for Sequential Code Parallelization," *Micro, IEEE*, July 2012.

[17] M. Lam and R. Wilson, "Limits of Control Flow On Parallelism," in *ISCA*. ACM Press, 1992, pp. 46–57.

[18] D. Jurafsky and J. H. Martin, *Speech & Language Processing*. Pearson Education India, 2000.

[19] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Comput. Linguist.*, vol. 18, no. 4, pp. 467–479, Dec. 1992. [Online]. Available: http://dl.acm.org/citation.cfm?id=176313.176316

[20] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944919.944937

[21] G. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numerische Mathematik*, vol. 14, no. 5, 1970.

[22] C. Lee, M. Potkonjak, and W. Mangione-Smith, "Mediabench: a tool for evaluating and synthesizing multimedia and communications systems," in *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*, Dec 1997, pp. 330–335.

[23] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The splash-2 programs," in *ISCA*, June 1995.