ApproxEigen: An Approximate Computing Technique for Large-Scale Eigen-Decomposition

Qian Zhang[†‡], Ye Tian[†], Ting Wang[†], Feng Yuan[†‡] and Qiang Xu[†‡]

[†]CUhk REliable Computing Laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

[‡]Shenzhen Research Institute, The Chinese University of Hong Kong
Email: {qzhang,tianye, twang, fyuan,qxu}@cse.cuhk.edu.hk

ABSTRACT

Recognition, Mining, and Synthesis (RMS) applications are expected to make up much of the computing workloads of the future. Many of these applications (e.g., recommender systems and search engine) are formulated as finding eigenvalues/vectors of large-scale matrices. These applications are inherently error-tolerant, and it is often unnecessary, sometimes even impossible, to calculate all the eigenpairs. Motivated by the above, in this work, we propose a novel approximate computing technique for large-scale eigen-decomposition, namely ApproxEigen*, wherein we focus on the practically-used Krylov subspace methods to find finite number of eigenpairs. With ApproxEigen, we provide a set of computation kernels with different levels of approximation for data pre-processing and solution finding, and conduct accuracy tuning under given quality constraints. Experimental results demonstrate that ApproxEigen is able to achieve significant energy-efficiency improvement while keeping high accuracy.*

1. INTRODUCTION

The emerging Recognition, Mining and Synthesis (RMS) applications are expected to account for a significant portion of computational resources in modern data center. Many RMS applications (e.g., data dimension reduction [1], recommender system [2], and search engine[3]) are formulated as large-scale eigen-decomposition problem, i.e., finding specific eigenvalues and eigenvectors on a sparse and high dimensional data set. As such computation demand grows rapidly with the increase of the data size, how to perform energy-efficient large-scale eigen-decomposition is of great interest to the industry.

Recently, approximate computing, being able to tradeoff computation quality (e.g., accuracy) and computational effort (e.g., energy) by exploiting the error-resilience properties of applications, has attracted lots of attention from both academia and industry recently [4, 5]. It is natural to apply approximate computing to RMS applications because these applications often process noisy data sets and/or involve a human interface with limited perceptual capability, and there is usually no specific "golden" output value that must be computed. In particular, for the above-mentioned eigen-decomposition problem, approximating the computations for those eigenvectors corresponding to smaller eigenvalues has little impact on the final solution quality. For example, in dimension reduction (e.g., Principle Component Analysis), data along the eigenvectors with small eigenvalues can be pruned without much impact on the total information.

Motivated by the above, in this work, we propose *ApproxEigen*, a novel approximate computing technique for large-scale eigen-decomposition, wherein we focus on Krylov subspace methods to find finite number of eigenvalues and eigenvectors for positive semi-definite symmetric matrices. The reasons are as follows: (i). the computational complexity of intuitive methods, such as QR decomposition, to find eigenvalues and eigenvectors is approximately $O(n^3)$, and the similarity transformations in these methods cause fill-ins which would destroy the sparsity and result in unaffordable memory overhead. Consequently, Krylov subspace methods are the main solution used in practice [6]; (ii). Most of the RMS applications are associated with positive semi-definite symmetric matrices [7].

ApproxEigen is comprised of two stages: *offline resilience identification* and *online solution finding*. To be specific, at offline stage, we divide the original algorithm into a set of computation kernels, which are then assigned with different quality constraints by conducting error-resilience identification. Next, at online stage, Krylov subspace is constructed followed by the eigenvalue/vector calculations. Based on the fact that eigenvectors corresponding to larger eigenvalues contain more information than those with smaller values, we perform online approximation tuning and employ lightweight checkers to satisfy given computation qualitie constraints. The main contributions of this work include:

- We propose to conduct runtime accuracy tuning for the large-scale eigen-decomposition problem widely used in error-resilient RMS applications, which is able to achieve significant energy savings under given computation quality constraints;
- We propose a novel lightweight quality checker to monitor the intermediate computation quality, which can be generally used to all matrix-vector multiplication dominated applications.

The remainder of this paper is organized as follows. Section 2 provides the relevant background and motivates this paper. Section 3 outlines our overall solution. Technical details and experimental results are then presented in Section 4 and Section 5, respectively. Finally, Section 6 concludes this paper.

2. PRELIMINARIES AND MOTIVATION

2.1 Preliminaries

2.1.1 Eigen-Decomposition

Suppose matrix A is of $n \times n$ dimension and the nonzero vector v lives in $\mathbb{R}^n$. Then if the matrix multiplies v yields a constant multiple of v, that is

$$Av = \lambda v$$

the scalar λ is called eigenvalue of A and v is called eigenvector corresponding to eigenvalue λ.

978-1-4673-8388-2/15/$31.00 ©2015 IEEE

Generally speaking, finding all eigenvalues and eigenvectors is to solve a polynomial with degree-n, which is quite computatinally expensive, and *Abel-Ruffini Theorem* shows there is no general, explicit and exact algebraic formula for the roots of a polynomial with degree 5 or above. Therefore, eigenvalues/vectors for large matrices can only be obtained by approximate numerical methods (i.e., iterative methods).

2.1.2 Large-Scale Eigen-Problems

In big data era, RMS applications (e.g. Google PageRank) easily face matrices with tens of thousands of dimensions, and the computation of its complete spectrum is out of the question with the $O(n^3)$ complexity. Fortunately, it often suffices to compute just a few eigenpairs.

Power method can be used to find principal eigenvector for large sparse matrices. In this method, we start from a randomized vector v, and then iteratively compute

$$v_{j+1} = \frac{Av_j}{\|Av_j\|}$$

, and v will finally converge to the eigenvector with the largest eigenvalue. Although power method is very effective, it does not utilize the information brought by middle-step computations, which can be used to find other eigenvalues and eigenvectors.

In practice, *Krylov subspace methods* are often used to find k eigenvalues and eigenvectors simultaneously. This method first build up Krylov subspace:

$$\mathcal{K}_m(A, x) = span\{x, Ax, A^2x, ..., A^{m-1}x\}, k < m \ll n$$

, where

$$\{x, Ax, A^2x, ..., A^{m-1}x\}$$

is called the Krylov sequence, and m is the largest allowable subspace dimension given by designers.

Generally speaking, Krylov sequence forms a basis for Krylov subspace but it is ill-conditioned. Individually normalizing each vector of the sequence will make all these vectors converge to the first eigenvector (i.e., reduced to power method). Therefore, the sequence needs better normalization.

To tackle this problem, *Lanczos algorithm* effectively and efficiently builds an orthonormal Krylov basis for *Hermitian matrices* (*Arnoldi algorithm* for non-Hermitian matrices). However, limited by the ill-initialization problem and/or insufficient machine precision, in Lanczos algorithm, once the current computed j vectors lose orthogonality, the algorithm will restart from a revised vector v'. Let $Q_m = [q_1, q_2, ...q_m]$ be orthonormal basis (by Lanczos algorithm) for Krylov subspace $\mathcal{K}_m$, where m is usually larger than the target number of eigenpairs k we want to find. Then finding m eigenpairs can be processed by the following steps:

- Project A onto $\mathcal{K}_m$ to get a tridiagonal matrix: $H = Q^T AQ$;
- Find eigenpairs of H: (λ, y);
- λ is called *Ritz value* and provides approximation for eigenvalue of A, y is the Ritz vector and $v = Qy$ provides approximation for eigenvectors of A

$$H = \begin{pmatrix} \alpha_1 & \beta_2 & & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{m-1} & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ 0 & & & & \beta_m & \alpha_m \end{pmatrix}.$$

As the projection H is a $m \times m$ tridiagonal matrix, methods like Givens rotation can eliminate the nonzero entries below diagonal. The computation complexity of finding all H's eigenpairs can be as small as $O(m^2)$.

2.2 Related Work and Motivation

In the literature, there are a few low-power solutions for specific eigen-decomposition algorithm. Liu *et al.* [8] proposed architecture designs for three kinds of eigen-decomposition algorithms on small matrices (from 3×3 up to 8×8). Wang and Zambreno [9] implemented an efficient FPGA-based double-precision floating-point architecture for eigen-decomposition, which can handle up to 2000×2000 matrices.

On the other hand, approximate computing (e.g., [10, 11]) has been advocated to trade off computation quality for energy savings and/or performance improvement. While a majority of existing approximate computing techniques were applied for image processing applications (e.g., [12, 13]), there are some recent research efforts to expand it to other areas. In particular, for matrix methods, Schaffner *et al.* [14] proposed an approximate computing technique for direct Cholesky-decompostion-based linear system solver, which is applicable to linear systems with some specific strutures.

Given that eigenvectors corresponding to smaller eigenvalues contain less information than those with larger values, intuitively, approximating these computations would have little impact on the final output quality. For example, for a face recognition application, Fig. 1 shows that the last several eigenvectors are nearly all noises. To the best of our knowledge, however, approximate computing for eigen-decomposition have not been explored in the literature, which motivates the proposed *ApproxEigen* technique in this work.

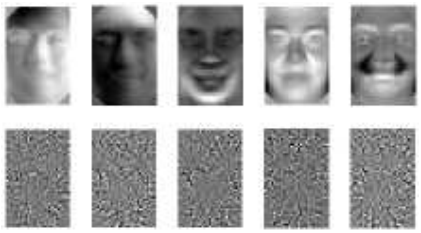


Figure 1: First Five and Last Five Eigenvectors for Face Recognition.

3. OVERVIEW OF APPROXEIGEN

Given a positive semidefinite symmetric matrix, ApproxEigen, running on quality-configurable platform with several different approximation modes, outputs a finite number of eigenvalues and eigenvectors. As shown in Fig. 2, ApproxEigen is comprised of two stages: offline resilience identification stage and online solution finding stage.

ApproxEigen can be conveniently integrated into any quality-configurable platforms (hardware or software). In this paper, we apply ApproxEigen onto the imprecise GPGPU framework presented in [15], based on CAD synthesis tools, GPGPU-Sim [16], and GPUWattch [17] simulation models to quickly evaluate the impact of ApproxEigen on the output quality and energy consumption of the approximate GPU system. It should be noted that our methodology can be easily ported to other platforms as well (see Appendix).

3.1 Offline Resilience Identification

As shown in Fig. 3, we first partition the Lanczos algorithm into several computation kernels, namely *subspace construction* (which includes *Krylov sequence calculation* and *subspace*

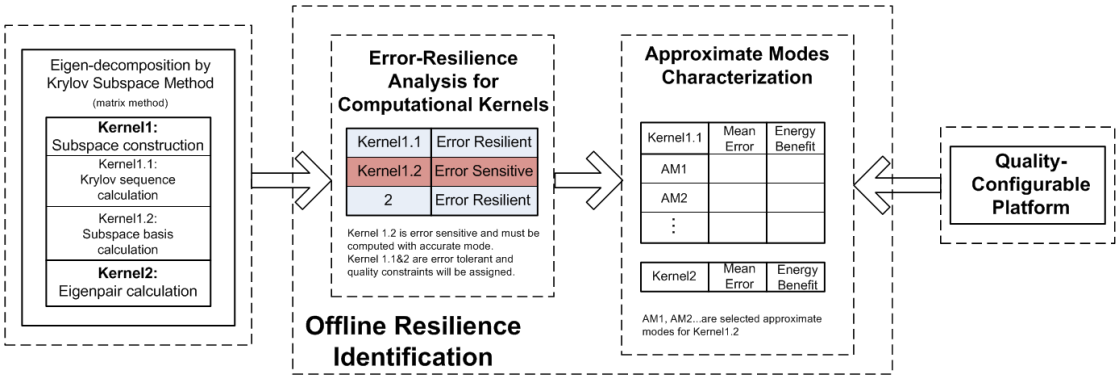


Figure 3: Offline Resilience Identification.

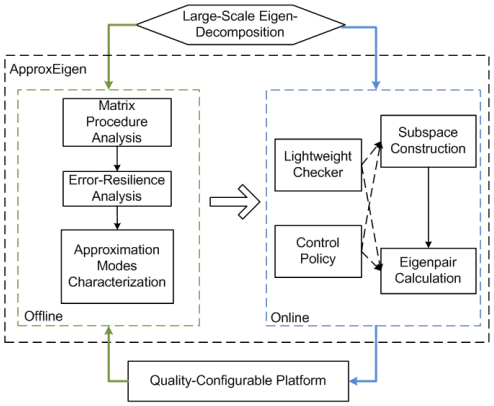


Figure 2: The Proposed ApproxEigen Technique.

basis calculation) and *eigenpair calculation*. Here one computation kernel is a well-formatted segment (e.g., one procedure performing the same operations iteratively) of the whole algorithm. Among these kernels, *subspace basis calculation* is error-sensitive as the loss of orthogonality with approximation is not preferred while the other two are error-resilient. Such identification is completed by using the error-injection based simulation platform proposed in [5].

Next, we conduct simulation on representative workloads for each of the resilient kernels with various approximations, then we could get the impact of these approximation modes, and obtain the *mean error* and *energy benefits* for each of them. These values would be sorted and used for online reconfiguration.

3.2 Online Solution Finding

At online stage, the algorithm iteratively reaches the final solution. We briefly describe the tasks in each computation kernel. Note that, the reconfiguration control policy of ApproxEigen include incremental accuracy tuning and selective restart for subspace construction, adpative approximation for eigenpair calculation, and rollback mechanism with the help of lightweight checkers. They are detailed in Section 4.

Subspace Construction: Given the dimension of Krylov subspace and a random vector, the Krylov sequence is first generated. This procedure itself involves restart (see Section 2.1.2) when the orthogonality of these vectors is lost, which can be reused when approximate computing is applied (as detailed in Section 4.1.1). Then, the algorithm will project the original matrix onto this constructed subspace to get a tridiagonal matrix with reduced dimension.

Eigenpair Calculation: With the projected tridiagonal matrix (see Section 2.1.2), we first reduce it to upper diagonal matrix in as few as $O(m)$ operations. Then, eigenvectors are computed using approximate arithmetic units with different accuracy levels. The accuracy tuning policy for this kernel is based on the following facts: (i). eigenvectors corresponding to larger eigenvalues keep more information and (ii). the first vector in Krylov sequence will converge to the principal eigenvector (i.e., power method), as discussed in Sec. 4.1.2.

4. RUNTIME ACCURACY TUNING UNDER QUALITY CONSTRAINTS

In this section, we detail how ApproxEigen performs runtime accuracy tuning and guarantees the computation quality for the reslient kernels used in it.

4.1 Accuracy Tuning for Computation Kernels

4.1.1 Subspace Construction

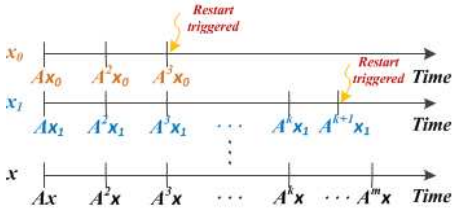
Given a random vector x, lanczos algorithm first computes Krylov sequence and then orthogonalizes these vectors to generate an orthonormal basis spanning the m-dimensional Krylov subspace corresponding to the input matrix A. It can be proved that with ideal arithmetic operations, eigenvalues/vectors solved by Krylov subspace with any initial vector are good approximations to those of the original matrix A [6].

However, as shown in Section 2.1.2, limited precision is likely to affect the numerical stability (e.g., orthogonality) of this algorithm unless the initial vector is well-conditioned. To find such a vector, Lanczos restarts to change the random vector x to x' based on the runtime information, and then repeat the whole procedure as shown in Fig. 4(a).

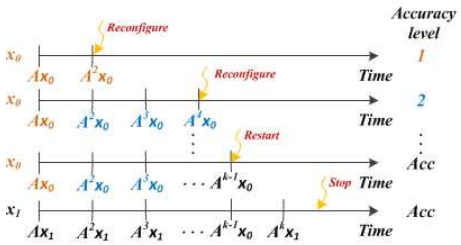
Suppose we want to find k eigenpairs. The algorithm requires m instead of exactly k ($m > k$) basis vectors for the constructed subspace (because larger dimension Krylov subspace has higher probability of holding good eigenvalue/vector approximations), the standard algorithm will restart without considering the position of the triggered restart. However, restart is only necessary when the loss of orthogonality happens within k vectors. Therefore, we take advantage of software-level *selective restarts* for energy efficiency, in which the restarts will be performed if and only if both of the following two conditions are satisfied:

- *Dimension Condition*: the number of vectors in the current basis is less than k;
- *Precision Condition*: the current computation mode is fully accurate.

Besides, each time the restart is triggered, the computation result becomes more accurate, therefore it is natural to



(a) Actual Procedure.



(b) Incremental Reconfiguration and Selective Restart.

Figure 4: Incremental Accuracy Tuning and Selective Restart.

gradually increase the accuracy levels of the arithmetic units. We adopt *incremental reconfiguration* for the hardware MAC units. To be specific, for a given randomized vector x, we start the Krylov sequence computing with the least accurate mode (obtained from offline characterzation, see Section 3.1). Once the restart operation is triggered, we reconfigure the computation mode to the next adjacent more accurate one until we get to the fully accurate mode. And the trigger mechanism is exactly the restart checking in original algorithm.

By utilizing selective restart and incremental reconfiguration in our subspace construction procedure, as shown in Fig. 4(b), we can get the constructed l-dimensional subspace $\mathcal{K}'$ ($k < l \leq m$). Then, we compute the projection of the original matrix on approximate $\mathcal{K}'$ instead of on accurate $\mathcal{K}$. By doing so, we can tradeoff the accuracy of eigenvalues/vectors to get better energy efficiency.

4.1.2 Eigenpair Calculation

With the projected approximate $l-dimensional$ tridiagonal matrix H', we can easily get all the eigenvalues by eliminating off-diagonal entries (e.g., givens rotation). And the magnitude of these eigenvalues can be naturally used to guide the accuracy tuning for computing eigenvectors, because eigenvector corresponding to larger eigenvalue keeps more information, and hence higher accuracy level is needed.

Suppose all the eigenvalues are denoted by $\lambda = [\lambda_1, \lambda_2, ...\lambda_l]^T$, where $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_l \geq 0$, we first define the following "relative importance" function, denoted by f_i, of each eigenvector.

Definition 1. For each eigenvector i,

$$f_i = \frac{\lambda_i}{\sum_i \lambda_i}$$

indicates its relative importance in terms of the ratio between its corresponding eigenvalue and the sum of all previous ones.

For any given i-th eigenvector, we have

$$f_{i+1} - f_i = \frac{\lambda_{i+1}}{\sum_{i+1} \lambda_{i+1}} - \frac{\lambda_i}{\sum_i \lambda_i} \leq 0 \ ,$$

because $\lambda_{i+1} \leq \lambda_i$ while $\sum_{i+1} \lambda_{i+1} \geq \sum_i \lambda_i$. This feature makes the relative importance function a monotonically decreasing function, which means the information percentage with each eigenvector is decreasing. Based on this fact, we can start eigenvector computing with fully accurate mode and gradually relax the accuracy requirement at runtime to get energy savings.

Here we define the "slope" of the function f (i.e., its change rate), as follows

$$s = \frac{f_{i-1} - f_i}{(i-1) - i} = f_{i-1} - f_i \ ,$$

to guide the hardware reconfiguration. If s is larger than a pre-defined threshold (e.g., $\tan 75^\circ$), the information carried by the current to-be-solved eigenvector is much less than its previous one, so lower accuracy level is preferred to achieve energy savings.

4.2 Rollback with Lightweight Quality Checking

The reconfiguration policy discussed earlier, while being able to effectively tradeoff accuracy and energy, cannot guarantee to meet a specified computation quality constraint. In Approx-Eigen, we resort to rollback computation to resolve this issue, with the help of a lightweight checker design used to evaluate whether the current accuracy level is sufficient or not.

It is very difficult, if not impossible, to design a light-weight checker at application-level because of the diversity of applications' characteristics. Previous works (e.g., [18]) often requires to run both the exact and the approximate kernels together for quality checking, which incurs significant energy overhead. Fortunately, lightweight quality checking is feasible for our problem as it is always possible to sample a subset of the matrix and use it to validate the solution quality. To be specific, in this paper, we propose a lightweight checker for general matrix-vector multiplications because they are the major operations used in our subspace construction computation kernel. As the eigenpair calculation kernel is working on a projection with relatively small matrices, accurate checking (e.g., angle between accurate vector and the approximate one) is affordable. We detail our proposed lightweight checker design as follows.

Consider a matrix-vector multiplication $Ax(= b)$. After getting the result vector b, we could easily check the quality to verify whether the residual $b' - b$ (where b' is the output from runtime checker) is within an acceptable range. In real life applications, however, the matrix A could be very large. If we use the residual directly as quality checker, the overhead would be unaffordable. Instead, we could implement a lightweight checker by verifying:

$$(c^T A)x = c^T(Ax)$$

Intuitively, the checker computes the projections of original Ax (i.e., b) and checker's Ax on the same vector c. If there are any computation errors, the two projections are unlikely to be equal. As shown in Eqn. 1, if $c = \overline{1}$, this computation is equivalent to computing the vector of A's column sums (i.e., $c^T A$) and multiplying it by x.

$$c^T Ax = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{nn} & \cdots & A_{nn} \end{pmatrix} \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix} \quad (1)$$

$$= \begin{pmatrix} \sum A_1^T & \sum A_2^T & \cdots & \sum A_n^T \end{pmatrix} \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix}$$

$$= \sum (x_i \sum A_i^T)$$

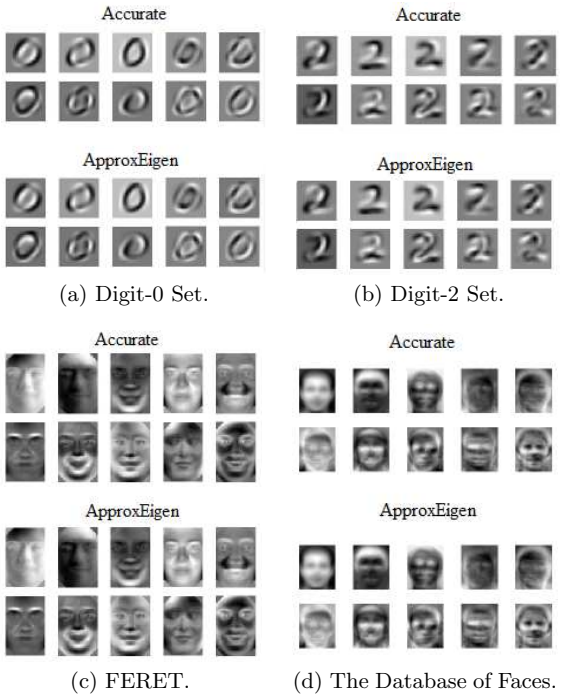
Table 1: Applications and Datasets

Application	Dataset	Source	Traing Samples	Testing Samples	Feature Dimension
Digit Recognition	Digit-0 set	[19]	6,000	1,000	400(20*20 image)
	Digit-2 set	[19]	6,000	1,000	400(20*20 image)
Face Recognition	FERET	[20]	990	700	1,107(41*27 image)
	The Database of Faces	[21]	400	400	10,304(112*92 image)

Thus, we could perform lightweight checking over a randomly sampled subset of columns, which can be easily implemented by setting c to a binary vector. Considering matrices may feature some specific characteristics, random sampling may not be always applicable. Consequently, we propose to categorize matrices into four categories and design lightweight checkers for them differently, as shown in the following.

- Random Checking: select A's columns randomly (i.e., make c a random vector) if the variance of A's column sums is low;
- Clustering Checking: perform clustering on A's columns and then randomly select representative one for each cluster if the variance of A's columns sums is high but these sums have banded structure;
- Identity-conditioning Checking: take c that satisfied $(c^T A)x = \overline{1}^T x = \sum x$ if the variance of A's column sums is high and these sums have no obvious structure;
- Null-conditioning Checking: take c that satisfied $(c^T A)x = 0$ if x is a high-variance vector.

By such sampling, the runtime evaluation can be done in $O(kn)$ (k is the number of columns to be sampled) setup time and $O(n)$ checking time, which is significantly cheaper than the original $O(n^2)$ time.



(a) Digit-0 Set. (b) Digit-2 Set.

(c) FERET. (d) The Database of Faces.

Figure 5: Comparison on Eigenvectors.

5. EXPERIMENTAL RESULTS

5.1 Experimental Setup

To evaluate the effectiveness of the proposed *ApproxEigen* technique, we perform simulation on two representative GPGPU applications with standard open-source datasets. Similar to [15], we use the imprecise FPUs and SFUs, whose "precise" counterparts are frequently used in computing-intensive applications and rank among the highest power consumptions in a GPU, to get the quality-energy tradeoff. Integer ALUs are not considered given their use in address calculations and their relatively small contribution to the total GPU power.

Table 1 describes the detailed information of these applications. The first one is a hand writing digit recognition application with two subsets from *The MNIST Database* [19], while the other is a face recognition application with two datasets. The download sources, number of samples, and feature dimensions for each dataset are also listed in Table 1. GPGPU-Sim simulations are performed on a RedHat Linux server with CUDA 4.0, gcc 4.4.7, and GPGPU-Sim 3.2.1 (integrated with GPUWattch).

5.2 Results of ApproxEigen

The recognition applications used in our experiments generate eigenpairs with training samples, and then use these eigenpairs to conduct recognition (i.e., classification) for the testing samples. In this subsection, we first show the differences on eigenpairs generated by accurate computing and ApproxEigen, followed by presenting the differences of classification results (i.e., the impact of ApproxEigen on applications) using these two groups of eigenpairs.

5.2.1 Quality and Energy

Our first experiment compares the eigenpairs calculated by accurate computing and ApproxEigen, and their corresponding energy consumptions.

Albeit many kinds of constraint settings are applicable, the kernel-based constraints are used in our experiments to perform fine-grained quality control. For each of these kernels, we use the "cosine distance" (between accurate vectors and approximate ones) as our error metric, and "fixed-interval calibration" in [18] to perform quality checking. Detailed constraint settings are generated from the off-line characterizations (on representative workloads) and listed in Table 3. Column "Application" indicates the constraint on application-level classification. And column "kernel 1.1" and "kernel 2" represent the constraint on each corresponding kernel.

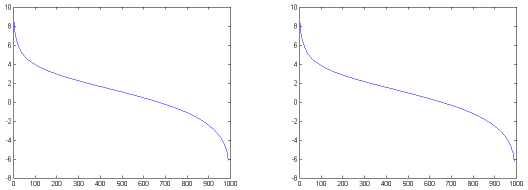
Table 3: Constraint Settings

Dataset	Constraint		
	Application	Kernel1.1	Kernel2
Digit-0 Set	0.5%	cos5°	cos3°
Digit-2 Set	0.5%	cos10°	cos5°
FERET	0.5%	cos15°	cos10°
The Database of Faces	0.5%	cos10°	cos5°

Fig. 5 presents the several principal eigenvectors calculated by fully accurate mode (first two lines in each subfigure) and ApproxEigen (last two lines in each subfigure), respectively. And the graphical differences can hardly be distinguished by human eyes. Similar results can be also found on log-eigenvalues in Fig. 6.

Table 2: ApproxEigen: Energy Savings

Application	Dataset	E. Savings (Arith.)	E. Savings (Holistic)
Digit Recognition	Digit-0 set	49.86%	22.53%
	Digit-2 set	89.54%	34.82%
Face Recognition	FERET	91.68%	49.83%
	The Database of Faces	55.59%	24.08%



(a) Accurate. (b) ApproxEigen.

Figure 6: Comparison on FERET LogEigenvalues.

Table 2 shows the results on energy savings of ApproxEigen (instead of whole application) for each dataset.Column "E. Savings (Arith.)" and column "E. Saving (Holistic)" give the energy saving percentages on arithmetic operations and the GPU system, respectively. It should be noted that the holistic energy consumptions reported by GPUWattch has taken the runtime checking overheads into consideration.

First of all, our results on four standard datasets show that ApproxEigen can bring significant 22.53% $\sim$ 49.83% holistic energy savings. This is not a surprising result because for eigen-decomposition, matrix-vector multiplication operations constitute the majority of runtime cycles in an linear algebra algorithm. Second, the energy savings vary significantly among different applications and datasets. We can get around 22.53% energy savings on "Digit-0 set" and close to 24.08% energy savings on "FERET". Third, arithmetic operations' energy savings and system-level energy savings are two different stories. Dataset "Digit-2 set" and "FERET" shares the similar energy savings on arithmetic units, however, the holistic energy savings are 34.82% and 49.83%, respectively. As the portion of energy consumed by FPUs and SFUs are similar among these datasets, such big difference is mainly caused by the specific characteristics of dataset and its initialization. In our methodology, besides arithmetic units, we adopt selective restart to get dramatic energy savings, and this part is highly dependent on the randomized initial vector for Krylov sequence. If the algorithm starts with a bad initialization vector (i.e., more restarts in the original procedure are needed), replacing the original restart by selectively reconfiguration and/or restart will report significant energy savings, as shown in "FERET".

Table 4: ApproxEigen:Impacts on Applications

Application	Dataset	Differences /Testing Samples
Digit Recognition	Digit-0 set	1/1000
	Digit-2 set	4/1000
Face Recognition	FERET	3/700
	The Database of Faces	0/400

5.2.2 *Energy Savings and Impacts on Applications*

Our second experiment is performed to check ApproxEigen's impacts on high-level applications, i.e., the classification results on testing samples.

In Table 4, Column "Differences/Testing Samples" indicates the number of wrongly classified samples (i.e., hamming distance) compared to the accurate classification results out of the total number of testing samples. We can see that only 4 testing samples are classified into wrong groups for "Digit-2 set", while we reach 100% correct classification result for "The Database of Faces". All of these testing samples demonstrate our generated eigenpairs only lead to less than 0.43% (i.e., 3/700 on "FERET") application-level quality loss (i.e., the proportion of wrongly classified samples in the whole testing ones), which is within our pre-defined constraint.

6. CONCLUSIONS

Large-scale eigen-decomposition is an energy-demanding task and has a wide range of applications, such as recommender systems and search engines. Motivated by the inherent error-resilience characteristics of these applications, in this work, we proposed an energy-efficient approximate computing technique to solve this problem, namely *ApproxEigen*, wherein we focus on the practically-used Krylov subspace methods to find finite number of eigenpairs. Our experiments on benchmark applications show that the proposed technique is able to achieve significant energy savings with negligible quality loss.

7. ACKNOWLEDGEMENT

This work was supported in part by the Hong Kong S.A.R. General Research Fund (GRF) under Grant No. 418112 and Grant No. N_CUHK444/12 and in part by National Natural Science Foundation of China under Grant No. 61432017.

8. REFERENCES

[1] I. Jolliffe, "Principal component analysis," *Wiley Online Library*, 2005.
[2] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.
[3] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *Computer networks and ISDN systems*, vol. 30, no. 1, pp. 107–117, 1998.
[4] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in Proc. *ETS*, 2013.
[5] V. K. Chippa el al. "Analysis and characterization of inherent application resilience for approximate computing," in Proc. *DAC*, pp. 113, 2013.
[6] D. S. Watkins, "The matrix eigenvalue problem: GR and Krylov subspace methods," in *SIAM*, 2007.
[7] Y.-K. Chen, et al. "Convergence of recognition, mining, and synthesis workloads and its implications," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 790–807, 2008.
[8] Y. Liu, et al. "Hardware efficient architectures for eigenvalue computation," in Proc. *DATE*, vol. 1, pp. 1–6, 2006.
[9] Y. Wang, et al., "An efficient architecture for floating-point eigenvalue decomposition," in Proc. *FCCM*, 2014.
[10] Q. Zhang, et al., "ApproxIt: An Approximate Computing Framework for Iterative Methods," in Proc. *DAC*, pp. 1–6, 2014.
[11] Q. Zhang, et al., "ApproxANN: an approximate computing framework for artificial neural network," in Proc. *DATE*, pp. 701–705, 2015.
[12] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in Proc. *DAC*, pp. 820–825, 2012.
[13] R. Ye, et al. "On reconfiguration-oriented approximate adder design and its application," in Proc. *ICCAD*, pp. 48–54, 2013.
[14] M. Schaffner, et al. "An approximate computing technique for reducing the complexity of a direct-solver for sparse linear systems in real-time video processing," in Proc. *DAC*, pp. 1–6, 2014.
[15] H. Zhang, M. Putic, and J. Lach, "Low power GPGPU computation with imprecise hardware," in Proc. *DAC*, pp. 1–6, 2014.

Table 5: ApproxEigen_Hardware: Energy Savings and Impacts on Applications

Application	Dataset	Differences /Testing Samples	E. Savings (Accumulator)
Digit Recognition	Digit-0 set	0/1000	21.54%
	Digit-2 set	3/1000	49.82%
Face Recognition	FERET	1/700	74.83%
	The Database of Faces	0/400	37.09%

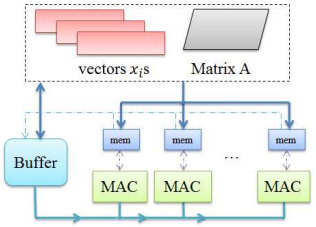
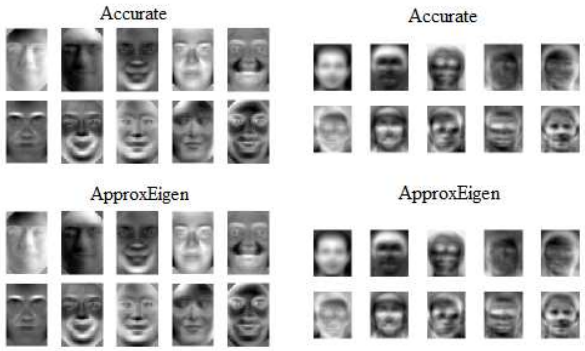


Figure 7: Hardware Architecture.



(a) FERET. (b) The Database of Faces.

Figure 8: Comparison on Eigenvectors.

[16] A. Bakhoda, et al. "Analyzing CUDA workloads using a detailed GPU simulator," in *ISPASS*, pp. 163–174, 2009.

[17] J. Leng, et al. "GPUWattch: enabling energy optimizations in GPGPUs," in Proc. *ISCA*, pp. 487–498, 2013.

[18] M. Samadi, et al. "Sage: Self-tuning approximation for graphics engines," in *Proc. of Micro*, pp. 13–24, ACM, 2013.

[19] Y. LeCun, C. Cortes, and C. J.C. Burges, "http://yann.lecun.com/exdb/mnist/".

[20] X. Wang and X. Tang, "A unified framework for subspace face recognition," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1222–1228, 2004.

[21] AT&T Laboratories and Cambridge University, "http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html".

[22] A. Majumdar, et al. "A Massively Parallel, Energy Efficient Programmable Accelerator for Learning and Classification," in *ACM Transactions on Architecture and Code Optimization*, vol. 9, pp. 6:1–6:30, 2012.

[23] E. J. King and E. Swartzlander, "Data-dependent truncation scheme for parallel multipliers," in *IEEE Asilomar Conference on Signals, Systems & Computers*, vol. 2, pp. 1178–1182, 1997.

9. APPENDIX

Our propsoed technique is also applicable to other approximate hardware designs and/or software implementations. In this appendix, we will show the results of ApproEigen on specific parallel hardware architecture.

9.1 Hardware Architecture

We resort to the architecture [22] shown in Fig. 7 as our hardware platform. It contains an array of multiplier-accumulator units (MACs) connected with its own local RAMs (*mem* in the figure), wherein quality-configurable arithmetic units are used in the MAC design. And each MAC only reads the corresponding vector from off-chip memory to its local memory. By streaming the multiplier vector through each MAC, matrix-vector multiplication is performed naturally (with each MAC computing the dot product of two vectors).

Table 6: Hardware Configurations (130nm, 1.2V)

Config.	Output Bits	Power(mW)	Power(%)
M_1	18	6.526	16.65
M_2	22	16.20	41.33
M_3	26	25.20	64.29
M_4	32	39.20	100

9.2 Approximate MAC Units

Given that nearly all of the computations in the proposed method is associated with matrix-vector multiplication, with-proposed in [23] as our arithmetic units. This multiplier has a tunable output of $(n + k)$ bits, where n represents the bit-width of inputs data. By using different values of parameter k, we can tradeoff computational accuracy and energy consumption. Detailed hardware configurations is presented in Table 6, where "Power(mW)", obtained by Synopsys PrimeTime, is the power on running dot product of two 10-entry vectors (i.e., the power of MAC with inexact multipliers), and "Power(%)" is the percentage of power value with respect to our fully accurate mode. Note that, we do not consider to use approximate adders in ApproxEigen due to their small energy impact.

9.3 Results of ApproxEigen

Quality and Energy.

Similar to the experiments on GPU platform, we first compare eigenparis computed by ApproxEigen with the accurate ones. It showes that the proposed *ApproxEigen* can also get notable case-by-case energy savings with minor quality loss on specific hardware platform.

Fig. 8 presents the graphical comparison on eigenvectors and human eyes can hardly distinguish the differences between the accurate ones and approximate ones.

After performaing simulation based on $45nm$ standard cell library with $1V$ supply voltage by commercial Synopsys EDA tools, we get the energy saving percentages listed in the right-most column "E. Savings (Accumulator)" of Table 5. We can get 21.5390% energy savings on "Digit-0 Set", but for "FERET", it can reach about "78.83%".

Impacts on Applications.

We then check ApproxEigen's impacts on high-level applications by comparing the classification results using accurate and our approximate eigenpairs based on the testing samples. Column "Differences/Testing Samples" in Table 5 indicates the number of wrongly classified samples compared to the accurate classification results out of the total number of testing samples. Our results on four standard datasets show that our generated eigenpairs only lead to less than 0.3% (i.e., 3/1000 for "Digit-2 set") quality loss.