TeraPCA: A fast and scalable method to study genetic variation In tera-scale genotypes

Aritra Bose[1]*, Vassilis Kalantzis[2]*, Eugenia Kontopoulou[1]*,
Mai Elkady[1], Peristera Paschou[3], Petros Drineas[1]

[1] Department of Computer Science, Purdue University, West Lafayette, IN
[2] Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN
[3] Department of Biological Sciences, Purdue University, West Lafayette, IN
*equal contribution

Motivation

- Advances in sequencing and genotyping technology leads to reduction in costs and availability of more data such as, recently a clustering of 777,000 individuals were carried out across North America [1]. Publicly available data sets such as UK Biobank, UK10K, GnomAd, etc. are making more data available.

- Principal Components Analysis (PCA) of genotypes is an established approach for population stratification and detecting substructure within populations. As modern datasets breach the TB level of size, "out-of-core" approaches are necessary.

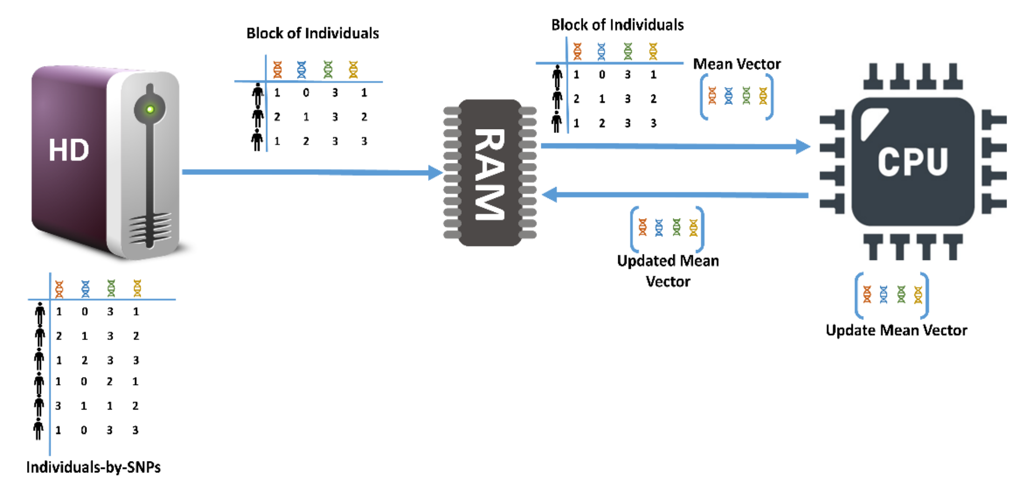
- Current state-of-the-art packages dealing with this problem are FastPCA[2] and flashPCA2[3]. flashPCA2 is a more recent approach which takes the advantage of the implicitly Restarted Arnoldi method implemented in the Spectra library[4]. Our goal is to design an approach that is equally fast (or faster) and shares no dependencies to external libraries.

- TeraPCA is a multi-threaded C++ library based on Intel's MKL library (or any other BLAS and/or LAPACK distribution). TeraPCA features no dependencies to external libraries and combines the robustness of subspace iteration with the power of randomization.

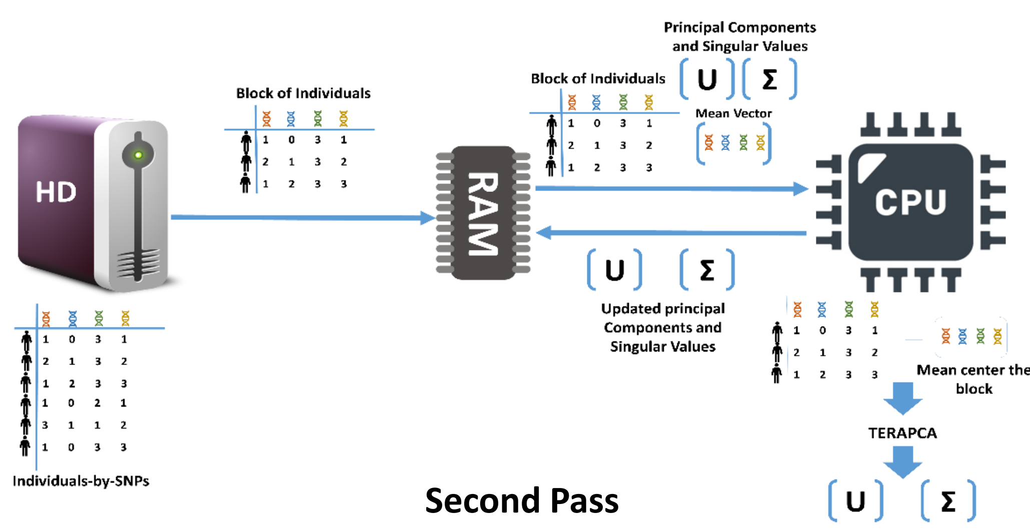
[1] Han et al. (2017), *Clustering of 770,000 genomes reveals post-colonial population structure of North America*, Nat. Comm. 8
[2] Galinsky et al. (2016), *Fast Principal-component analysis reveals convergent evolution of ADH1B in Europe and East Asia*. Am. J. Hum. Genet., 98, 456-472
[3] Abraham et al. (2017), *FlashPCA2: principal component analysis of Biobank-scale genotype datasets*, Bioinformatics, Volume 33, Issue 17, September 2017, 2776-2778
[4] C++ library Spectra (http://yixuan.cos.name/spectra).

Method

TeraPCA is an **out-of-core** implementation of the randomized **subspace iteration method**. It is a **two pass** procedure where a) the mean vector is computed separately, and b) subspace iteration is applied to the data matrix. For large datasets, each block of rows is fetched and demeaned independently of each other.



First Pass



Second Pass

The approximate singular vectors are computed by applying SVD on a lower-dimensional subspace defined by

$$\widehat{U} = (AA')^q AX$$

where, $A \in \mathbb{R}^{m\,x\,n}$ and a $X \in \mathbb{R}^{n\,x\,r}$ is a matrix with i.i.d entries and **r** right hand side (rhs) columns.

Memory and Running Time

We ran our algorithm on a replicated matrix (from HGDP) of size 33,376 individuals and 2.65 mil. SNPs. We select block size of 100 rows and rhs of 30, 60, 90 and 300. For convergence criterion we used the singular value partial sum change between sequential iterations to be $\leq 10^{-3}$, i,e. at least 3 digits of accuracy.

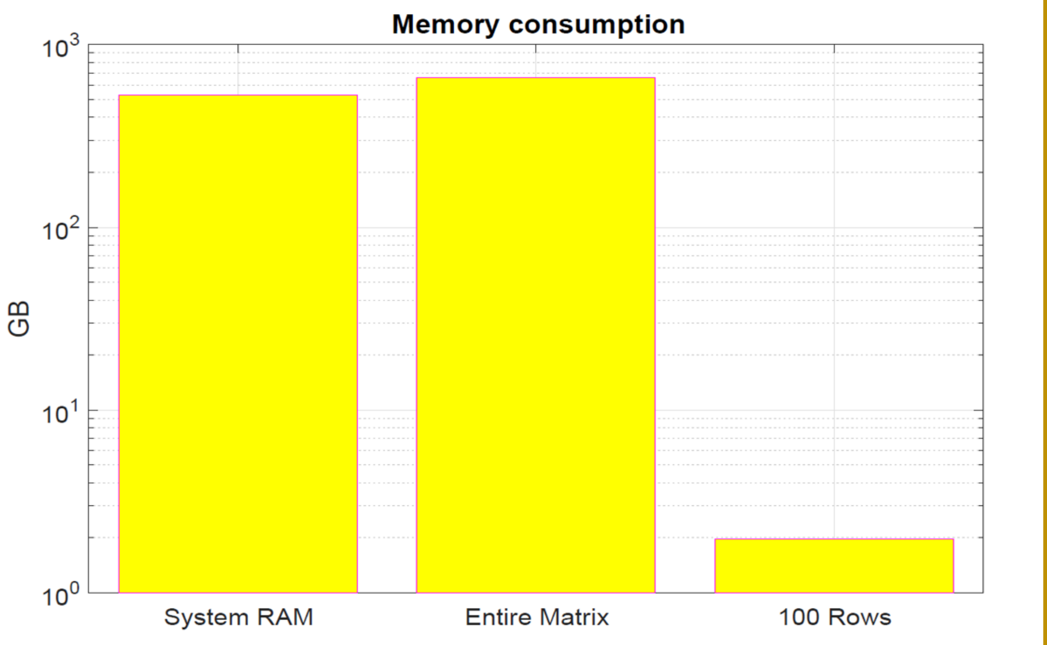


Fig 1. Difference in memory used to store the above matrix shows the benefit of using out-of-core algorithms

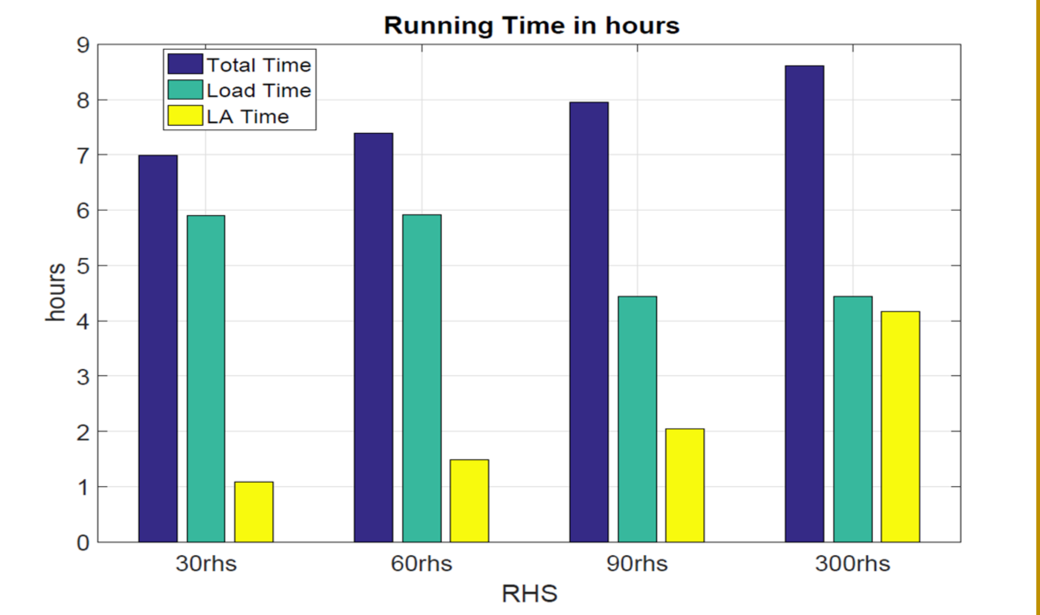
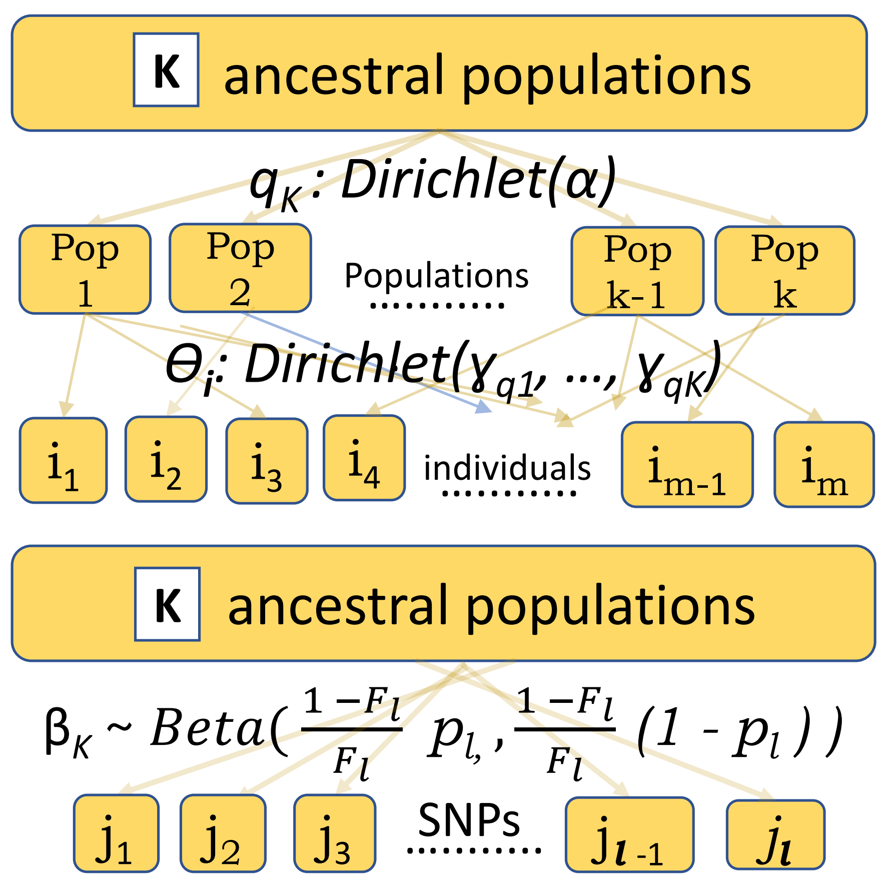


Fig 1. More rhs means more computation time but faster convergence.

Simulated Data sets

We ran TeraPCA on simulated data sets, which were generated using the Pritchard-Stephens-Donelly[5] (PSD) model, similar to the one generated in the package TeraStructure [6].



F_l : Wright's F_{st} at SNP location ℓ }
p_l : *marginal allele frequency* } Obtained from HGDP

Assume each SNP genotype $\boldsymbol{\ell}$ in each individual i denoted by $x_{\mathbf{i},\ell} \sim$ **Binomial(2,**$p_{\mathbf{i},\ell}$**)** where

$$\boldsymbol{p_{i,l}} = \textstyle\sum_k \boldsymbol{\theta_{i,k}\,\beta_{k,l}}$$

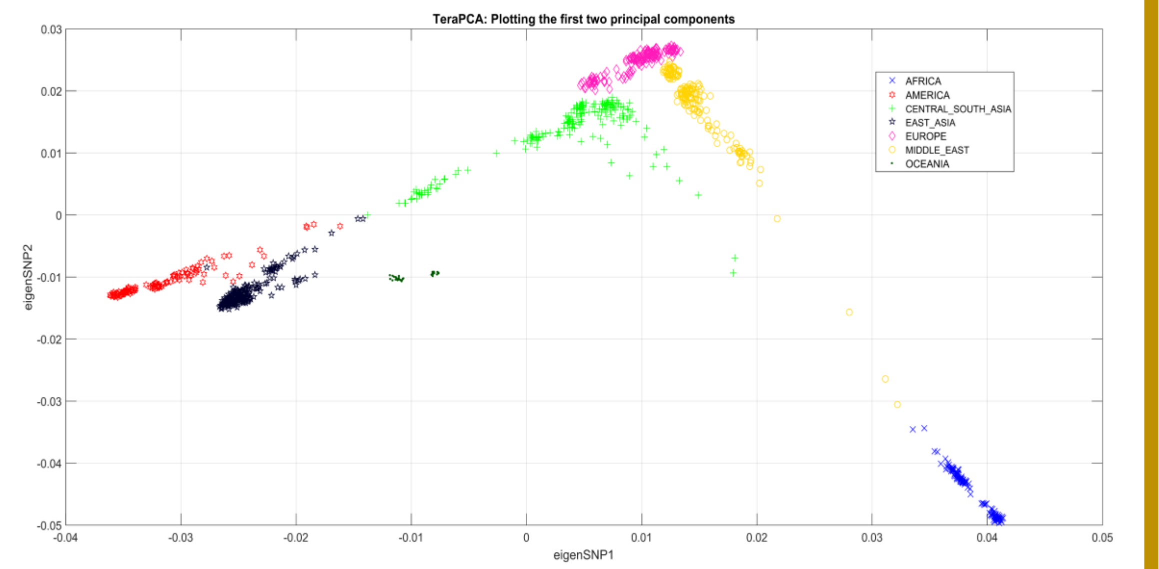
This returns a code for each SNP genotype as 0, 1, or 2 (to denote the three possible genotypes)

*The β's are computed by Balding-Nichols Model

[5] Pritchard et al. (2000), *Inference of population structure using multilocus genotype data*, Genetics, 155, 945-959
[6] Gopalan et al. (2016), *Scaling probabilistic models of genetic variation to millions of humans*. Nat. Genet., 48, 1587-1590

Qualitative Results

To evaluate the quality of our method, we replicated the HGDP data set to create data set containing 2K individuals and 1.4 mil. SNPs. TeraPCA computes the top 10 Principal Components (PCs) in **310 secs** in comparison to **385 secs** taken by flashPCA2 and FastPCA's **785 secs** (approximately). Plotting the top 2 principal components we find qualitatively TeraPCA performs the same as EIGENSOFT's smartpca fastmode.



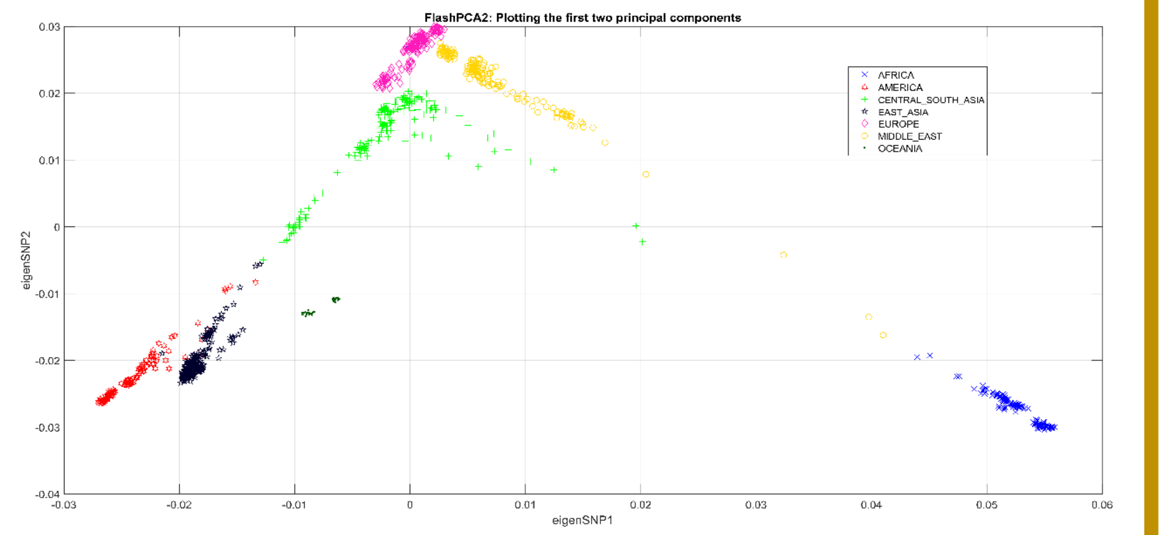


Fig 3. Plotting the top two PCs shows qualitative similarity between TeraPCA and FlashPCA2. EIGENSOFT's smartpca also had similar plots.

Quantitative Results

We compared the performance of **TeraPCA with flashPCA2**, as it performs the best out of the available packages. We used both real and simulated data sets to show that **TeraPCA performs better than flashPCA2** in most of the cases (marked in red) for varying number of threads.

Data sets	Size	Dimensions
Ethiopia (Pagani 2012)	939 MB	235 individuals, 1,047,265 markers
India (Basu 2015)	1.1 GB	367 individuals, 803,570 markers
HGDP (Li 2002)	2.6 GB	1043 individuals, 660,734 markers
Merged HGDP	11 GB	2085 individuals, 1,321,468 markers
100K -by- 100K (Simulated)	38 GB	100,000 individuals, 100,000 markers
1M -by- 100K (Simulated)	373 GB	1,000,000 individuals, 100,000 markers

Table 1. Data sets used in the analysis

	1 Thread		4 Threads		8 Threads	
Data sets	fPCA2	T-PCA	fPCA2	T-PCA	fPCA2	T-PCA
Ethiopia	49	80	53	61	47	32
India	58	60	63	32	54	17
HGDP	70	124	71	80	67	51
Merged HGDP	380	310	290	169	274	98
100K -by- 100K	26.38* mins	26 mins	26.25* mins	16 mins	27* mins	11 mins
1M -by- 100K	188 mins	159 mins	201.2 mins	101 mins	177.8 mins	62 mins

Table 2. Comparison between flashPCA2 (fPCA2) and TeraPCA (T-PCA). Time is in seconds, unless otherwise mentioned.

*times with loading the entire matrix into RAM, required 8GB of RAM. Without loading it takes approximately 47.4 mins.

(All computations were done in a two 10-Core Intel Xeon-E5 processor with 512 GB RAM)

Acknowledgements

This work was supported by NSF BigData 1661760, NSF IIS 1319280 and NSF IIS 1302231.