Asynchronous kernel ridgeless regression

Vijay Giri, Mikhail Belkin, Parthe Pandit

December 3, 2022

Abstract

Kernel machines are a class of predictors commonly used in machine learning. We propose a parallel and completely lock-free asynchronous algorithm, AsyncEigenPro, for kernel regression. This algorithm resembles Hogwild! but uses the special structure of the kernel regression problem. The main application of the algorithm is to enable efficient multi-GPU training for kernel methods. We show theoretically that the effect of delayed gradients and inconsistent reads on the rate of convergence can be minimal. We run large scale experiments to show near linear speedup in training time with respect to the number of GPUs.

1 Introduction

We are given a set of data points $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \in \mathbb{R}^d \times \mathbb{R}$, where $\boldsymbol{y} = (y_i) \in \mathbb{R}^n$ is the vector of targets. We consider an RKHS $\mathcal{H}$ corresponding to a positive definite kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and find the minimum norm interpolator

$$\widehat{f} = \underset{f}{\operatorname{argmin}} \ \|f\|_{\mathcal{H}} \qquad \text{subject to } f(\boldsymbol{x}_i) = y_i \quad \forall (\boldsymbol{x}_i, y_i) \in \mathcal{D}.$$

One can show that this estimator is equivalent to

$$\mathcal{L}(f) := \frac{1}{2n} \sum_{i=1}^{n} \left(f(\boldsymbol{x}_i) - y_i\right)^2, \quad f \in \mathcal{H} \tag{1}$$

$$f^* = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \ \mathcal{L}(f) + \tfrac{\lambda}{2} \|f\|_{\mathcal{H}}^2 \tag{2}$$

in the limit when $\lambda = 0^+$. Due to the representer theorem [5] and [16] we know that

$$f^* = \sum_{i=1}^{n} \alpha_i^* k(\boldsymbol{x}_i, \cdot) \tag{3}$$

where $\boldsymbol{\alpha}^* = (\alpha_i^*) \in \mathbb{R}^n$ is the solution to the linear system of equations

$$(\mathbf{K} + \lambda I_n)\boldsymbol{\alpha}^* = \boldsymbol{y} \tag{4}$$

where $\boldsymbol{y} = (y_i) \in \mathbb{R}^n$, and where $\mathbf{K}$ is the $n \times n$ kernel matrix $\mathbf{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

1.1 Main contributions

This paper provides an algorithm with theory and numerical experments validating the theory. Our code is available at [1].

AsyncEigenPro: We provide a new algorithm for performing kernel regression in a distributed and asynchronous manner across multiple GPUs.

Table 1: Works on synchronous and asynchronous SGD

	Synchronous SGD	Asynchronous SGD
Linear regression	[3]	[13, 10]
Kernel interpolation	[7]	This paper

Convergence analysis: We analyze convergence properties of our algorithm and study the effect of the delay in the read-write procedure on the underlying hyperparemters such as learning rate and batch size

Numerical experiments: We show kernel regression experiments on datasets of size upto 3 million, and show a speed-up compared to its synchronous counterpart.

1.2 Prior work

Distributed kernel regression: Distributed kernel regression has also been studied using the random features approximation in [6]. There have been several other approaches to solving the kernel regression problem given in equation (4) in a distributed manner, (see [11]). However these algorithms rely on a matrix inverse which makes them (i) unable to scale to large models, and (ii) unstable for $\lambda = 0^+$. Distributed kernel regression has been considered in [17] and several follow-ups. However our training algorithm does not explicitly approximate the kernel model.

Asynchronous SGD analysis with delays: Hogwild! [13] is a well-known algorithm for running stochastic gradient descent (SGD) with asynchronous updates from multiple processors into a single parameter vector in shared memory. This algorithm is particularly useful when SGD iterates are sparse. It's analysis is further simplified using a perturbed iterate framework [10]. We use some notions from these works. Generic analysis for SGD under delayed updates and slightly different environment has also been considered in [1].

EigenPro(EP): [8, 9] is an iterative solver for kernel regression using a preconditioned version of the Richardson iteration [14] to solve equation equation (4). EP applies two stochastic approximations – (i) a batching, which leads to block-coordinate updates, and (ii) Nyström approximation for the preconditioner. This makes the algorithm scalable to very large datasets. Hyperparameter tuning for EP is also well understood for the kernel ridgeless regression problem ($\lambda = 0^+$) due to [7].

2 EigenPro and friends

2.1 Background on EigenPro

EigenPro [8] uses preconditioned mini-batch stochastic gradient descent to efficiently solve (2). It is further improved [9] using Nyström extension for construction of preconditioner. Final result is a two step update to $\boldsymbol{\alpha}$ in each iteration which looks like follows

$$\boldsymbol{\alpha}^{\mathsf{batch}} \leftarrow \boldsymbol{\alpha}^{\mathsf{batch}} - \eta \boldsymbol{g}^{\mathsf{batch}} \tag{5}$$

$$\boldsymbol{\alpha}^{\mathsf{nys}} \leftarrow \boldsymbol{\alpha}^{\mathsf{nys}} + \eta \boldsymbol{M}_s \mathbf{K}^{(\mathsf{nys},\mathsf{batch})} \boldsymbol{g}^{\mathsf{batch}} \tag{6}$$

$$\text{where} \quad \boldsymbol{g}^{\mathsf{batch}} = \frac{1}{m}\left(\mathbf{K}^{(\mathsf{batch},n)}\boldsymbol{\alpha} - \boldsymbol{y}^{\mathsf{batch}}\right) \tag{7}$$

$$\boldsymbol{M}_s = \boldsymbol{E}\Lambda^{-1}\left(\boldsymbol{I} - s\lambda_{q+1}^{(s)}\Lambda^{-1}\right)\boldsymbol{E}^\top \tag{8}$$

Here, m is the mini-batch size and batch corresponds to mini-batch indices and nys corresponds to Nyström sample indices. $\mathbf{K}^{(\mathsf{batch},n)} \in \mathbb{R}^{m\times n}$ is the rectangular kernel matrix where m rows, corresponding to the m

[1] github

mini-batch indices, are selected from $\mathbf{K}$ in each iteration. Similarly, $\boldsymbol{\alpha}^{\mathsf{batch}}, \boldsymbol{y}^{\mathsf{batch}}$ corresponds to the vector's m dimensions corresponding to the mini-batch indices.

s samples for Nyström extension are randomly sampled once during preprocessing. $\mathbf{K}^{(\mathsf{nys},\mathsf{batch})} \in \mathbb{R}^{s\times m}$ is a matrix where s rows, corresponding to Nyström sample indices, are chosen with m columns corresponding to mini-batch indices. $\boldsymbol{E} \in \mathbb{R}^{s\times q}$ is a matrix which contains top-q eigenvectors of $\mathbf{K}^{(\mathsf{nys},\mathsf{nys})} \in \mathbb{R}^{s\times s}$ column-wise and Λ is a diagonal matrix which contains top-q eigenvalues of $\mathbf{K}^{(\mathsf{nys},\mathsf{nys})}$. $s\lambda_i^{(s)}$ is the i^{th} largest eigenvalue of $\boldsymbol{K}^{(\mathsf{nys},\mathsf{nys})}$. Eigenvalues are denoted with a superscript (s) to indicate Nyström approximation with s samples.

2.2 Distributed EigenPro with multiple preconditioners

We use 'processor' as a general term throughout the paper but the description holds for any computing construct like threads, GPUs, CPUs and processes. We can extend EigenPro in section 2.1 to utilize multiple processors by distributing the kernel and gradient computations. In each iteration the main processor can distribute, to each of the other processor, a part of kernel computation and the corresponding gradient computation. Then the main processor can gather these results and merge them to finally update the parameter vector. Since we have multiple processors, without any additional cost, we can have multiple preconditioners which results in low variance in Nyström approximation.

During preprocessing, each processor constructs a preconditioner with different Nyström samples $\mathsf{nys}_1, \mathsf{nys}_2, \ldots, \mathsf{nys}_G$. At each iteration we partition the batch into equal parts $\{\mathsf{batch}_r\}_{r=1}^G$ for G number of processors. Then, each processor can compute $\boldsymbol{g}^{\mathsf{batch}_r}$ as in (7) and $\boldsymbol{M}_{s_r}\boldsymbol{K}^{(\mathsf{nys}_r,\mathsf{batch}_r)}\boldsymbol{g}^{\mathsf{batch}_r}$. Finally, the main processor can gather these vectors to update the parameter as follows

$$\boldsymbol{\alpha}^{\mathsf{batch}} \leftarrow \boldsymbol{\alpha}^{\mathsf{batch}} - \eta[\boldsymbol{g}_{\mathsf{batch}_1}^\top, \boldsymbol{g}_{\mathsf{batch}_2}^\top, \ldots, \boldsymbol{g}_{\mathsf{batch}_G}^\top]^\top$$
$$\boldsymbol{\alpha}^{\mathsf{nys}} \leftarrow \boldsymbol{\alpha}^{\mathsf{nys}} + \eta \sum_{r=1}^{G} \boldsymbol{M}_{s_r}\mathbf{K}^{(\mathsf{nys}_r,\mathsf{batch}_r)}\boldsymbol{g}^{\mathsf{batch}_r}$$

This synchronous algorithm naturally has some disadvantages in cases when there are delays in computations. The main processor has to wait for each processor to complete the computation before proceeding to next iteration. As the number of processors increase this wait time also increases. See Figure 1 for comparison. Hence, we explore for asynchronous algorithm.

3 Asynchronous EigenPro

We propose AsyncEigenPro where each processor runs EigenPro in an asynchronous lock-free manner without any dependence on other processors. However, all processors update the parameter $\boldsymbol{\alpha}$ in a shared memory after every iteration and also read $\boldsymbol{\alpha}$ from the shared memory at the beginning of each iteration. We emphasize that there is no locking or synchronization at any point. We allow for inconsistent reads and as we will see further, the updates from each processors are completely non-overlapping. Hence, unlike prior works, we do not need to assume atomic writes even for single dimension. We explain these in detail.

We randomly partition the data into mutually exclusive and collectively exhaustive sets $\{\mathcal{D}_r\}_{r=1}^G$, one for each processor. EigenPro iteration entails sampling a batch of data at each iteration and a batch of data for Nyström approximation to construct preconditioner during preprocessing. In AysncEigenPro, each processor samples from its corresponding set $\mathcal{D}_r$ for both Nyström samples during preprocessing and mini-batch samples during each iteration. So, unlike EigenPro, here we have a different preconditioner constructed from disjoint Nyström samples for each processor. Due to this, note that from (5), (6) each processor r updates only indices belonging to $\mathcal{D}_r$.

The parameter vector $\boldsymbol{\alpha}$ is in a shared memory which every processor can access. Each processor would need the whole parameter vector to calculate gradients. Since each processor is updating independent of other processors, the value of the the parameter during read may be different from the value of parameter at the time of update because other processors might have updated the parameter. So, the gradient updates may not correspond to the current parameter (stale gradients). Also, while a processor is reading the parameter other processors might have partially updated the parameter which leads inconsistencies while reading the

parameter. We call this event inconsistent reads. We use notions from [10] to encapsulate both these events as perturbations to the parameters and denote it as $\widehat{\boldsymbol{\alpha}}$.

We restate the update equations in context of a processor r to indicate different preconditioner and $\widehat{\boldsymbol{\alpha}}$

$$\boldsymbol{\alpha}^{\mathsf{batch}_r} \leftarrow \boldsymbol{\alpha}^{\mathsf{batch}_r} - \eta \boldsymbol{g}^{\mathsf{batch}_r} \tag{9}$$

$$\boldsymbol{\alpha}^{\mathsf{nys}_r} \leftarrow \boldsymbol{\alpha}^{\mathsf{nys}_r} + \eta \boldsymbol{M}_{s_r} \mathbf{K}^{(\mathsf{nys}_r, \mathsf{batch}_r)} \boldsymbol{g}^{\mathsf{batch}_r} \tag{10}$$

$$\text{where} \quad \boldsymbol{g}^{\mathsf{batch}_r} = \frac{1}{m}\left(\mathbf{K}^{(\mathsf{batch}_r, n)}\widehat{\boldsymbol{\alpha}} - \boldsymbol{y}^{\mathsf{batch}_r}\right) \tag{11}$$

$$\boldsymbol{M}_{s_r} = \boldsymbol{E}_r \Lambda_r^{-1}\left(\boldsymbol{I} - s\lambda_{q+1}^{(s_r)}\Lambda_r^{-1}\right)\boldsymbol{E}_r^\top \tag{12}$$

Note that $\boldsymbol{M}_{s_r}$ is calculated using top-q eigensystem of $\mathbf{K}^{(\mathsf{nys}_r, \mathsf{nys}_r)}$.

For pseudo code see Algorithm 1. Note that none of the updates in Algorithm 1 need to be atomic

Algorithm 1 Async EigenPro

Require: Data $\mathcal{D}$ and G number of GPUs

$\boldsymbol{\alpha} \leftarrow$ Shared parameter vector partitioned as $(\boldsymbol{\alpha}_1, \dots \boldsymbol{\alpha}_G)$ for write permissions
`Setup`
for $r \leftarrow 1, ..., G$ **do in parallel**
    $\mathcal{D}_r^{\mathsf{nys}} \leftarrow$ randomly sample s points from $\mathcal{D}_r$
    $\boldsymbol{M}_{s_r} \leftarrow$ Compute preconditioner from $\mathcal{D}_r^{\mathsf{nys}}$ using equation (12)
end for

`Iteration`
for $r \leftarrow 1, ..., G$ **do in parallel**
    for $t \leftarrow 1, 2, ...$ **do**
        $\mathcal{D}_r^{\mathsf{batch}} \leftarrow$ randomly sample mini-batch of size m from $\mathcal{D}_r$
        $\widehat{\boldsymbol{\alpha}} \leftarrow$ read $\boldsymbol{\alpha}$ from shared memory
        $\boldsymbol{\alpha} \leftarrow$ update from a preconditioned gradient step using $(\boldsymbol{M}_{s_r}, \mathcal{D}_r)$ using equation (9), (10)
    end for
end for

4 Convergence of Async-EigenPro

To account for delayed gradients in analysis we define the following

Definition 1 (Iteration latency τ)**.** We assign a time t to an iteration when the update of the shared parameter starts. Between the time of read and time of update of iteration t, other processors might have updated to the shared parameter vector. We assume there exists a constant τ such that the maximum number of updates between time of read and time of update is no more than τ.

Let's define a $\mathbb{R}^{n\times n}$ version of $\boldsymbol{M}_s$ defined in equation (8) where we assign 0 to all the entries which do not correspond to Nyström indices.

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_s & \mathbf{0} \\ \mathbf{0} & \mathbf{0}_{n-s} \end{bmatrix} \in \mathbb{R}^{n\times n}, \ \boldsymbol{M}_s \in \mathbb{R}^{s\times s} \tag{13}$$

Define

$$\boldsymbol{P} := \boldsymbol{I} - \boldsymbol{M}\boldsymbol{K}$$

Before stating our result, we recall some notations. $\|\boldsymbol{x}\|_{\boldsymbol{M}}^2 = \boldsymbol{x}^\top \boldsymbol{M}\boldsymbol{x}$, m is mini-batch size per processor, $\lambda_i^{(s)}$ is the i^{th} largest eigenvalue of the scaled kernel matrix $\boldsymbol{K}/n$ approximated by s Nyström samples.

Table 2: Comparing convergence rates

	Synchronous SGD	Asynchronous SGD
Linear regression	[3] (rate)	[13, 10] (rate)
Kernel interpolation	[7] (rate)	This paper (rate)

Theorem 1 (Convergence of $\boldsymbol{\alpha}$)**.** *Consider the iteration given in equation* (9) (10) *initialized at* $\boldsymbol{\alpha}_0$*, with iteration latency* τ*. If the learning rate* $\eta \leq \frac{m}{\beta + \lambda_{q+1}^{(s)}(m-1)}$*, where* $\beta = \max_i k(\boldsymbol{x}_i, \boldsymbol{x}_i)$*, then after* T *iterations, we have*

$$\mathbb{E}\left\|\boldsymbol{\alpha}_T - \boldsymbol{\alpha}^*\right\|_{\boldsymbol{P}^{-1}}^2 \leq \left(1 - \tfrac{\eta\lambda_n}{2}\right)^T \left\|\boldsymbol{\alpha}_0 - \boldsymbol{\alpha}^*\right\|_{\boldsymbol{P}^{-1}}^2 + \frac{2\eta^2\tau\mathcal{L}_{\mathsf{init}}}{m}\left(1 - \left(1 - \tfrac{\eta\lambda_n}{2}\right)^T\right) \tag{14}$$

where $\mathcal{L}_{\mathsf{init}}$ *is the square loss* (1) *at initialization.*

Proof in Appendix A.1.

Remark 1 (Behaviour with delay τ)**.** The first term is an exponential convergence term since we are dealing with a linear dynamical system. The second term is a variance term which is introduced because of the delay τ. Notice that if $\tau = 0$, we are in the synchronous setting, in which case the result matches the parameter bound in [7, Thm. 1]. However, when $\tau > 0$, notice that even with interpolation we get a variance term due to delayed gradients. This leads to a different hyperparameter selection for the learning rate η and batch size m. These are discussed in detail in section 4.1

Remark 2. Comparing with Hogwild!, this analysis uses two aspects of the kernel regression problem at hand. The first that we can interpolate, the second that we have non-overlapping writes, which simplifies analysis. We also note that, unlike Hogwild!, there is no quadratic dependence on τ

4.1 Automatic hyperparameter tuning

We tune the batch size and learning rate under conditions of delay. We also consider a delay model which tells us how the behaviour of the algorithm must change with increasing the number of GPUs.

To get to a specified error ϵ we can make each of the two terms $\epsilon/2$ which results in following corollaries

Corollary 1. *Error rate of* ϵ *is reached i.e.,* $\left\|\boldsymbol{\alpha}_T - \boldsymbol{\alpha}^*\right\|_2^2 \leq \epsilon$ *after* T *iterations given by*

$$T \geq \mathcal{O}\left(\sqrt{\frac{\tau\mathcal{L}_0}{m\epsilon}}\frac{\log\left(\frac{2b_0}{\epsilon}\right)}{\lambda_n}\right)$$

Corollary 2. *To reach an error rate* ϵ *i.e.,* $\left\|\boldsymbol{\alpha}_T - \boldsymbol{\alpha}^*\right\|_2^2 \leq \epsilon$ *the maximum delay in gradients* τ *has an upper bound as follows*

$$\tau \leq \mathcal{O}\left(\frac{m\epsilon}{\eta^2\mathcal{L}_0}\right)$$

... more discussion on these soon

4.2 Effect of increasing number of processors

Remark 3 (Challenges in the interpolation regime)**.** If number of processors is p then $\tau = \mathcal{O}(p)$. We see from Corollary 1 that as we increase the number of processors, the number of iterations increase by $\sqrt{p}$. So, the theoretical guarantee is for $\sqrt{p}$ speedup. In interpolation regime, synchronous SGD has strong convergence properties due to no variance. Hence, introducing a variance due to delayed gradients results in less than linear speedup guarantee. However, in experiments we see that linear speedup is obtained for many datasets.

... more discussion on this soon

5 Numerical experiments

We run experiments on different datasets with both synchronous and asynchronous EigenPro. We also run experiments with delay models on each GPU. We run experiments on clusters upto 8 NVIDIA A100 GPUs.

Datasets: CIFAR10-3M is a 3 million subset of CIFAR10-like images from [12]. We generate a subset of ImageNet dataset using 9 classes belonging to living animals [4]. Then, we perform random augmentations to get a dataset of size 1.9 million images which we call ImageNet9 here. All image datasets are featurized using MobileNetV2 model [15] and then used as input for kernel regression. HIGGS data is from [2].

All comparisons are done such that the training loss reaches the same value in all cases of a dataset. Figure 1 shows the comparison between synchronous and asynchronous training. We assume a delay model where at each iteration each GPU has the same probability of delay. So a GPU can randomly delay gradient computation by a given probability. The effect of delay is distributed across iterations in the case of asynchronous where as the delay gets added to each iteration in synchronous case. Thus we see that as the probability of delay increases, the gap between synchronous and asynchronous becomes drastic.

...more discussion soon [The effect of delay as the number of processors increase with same delay probability]

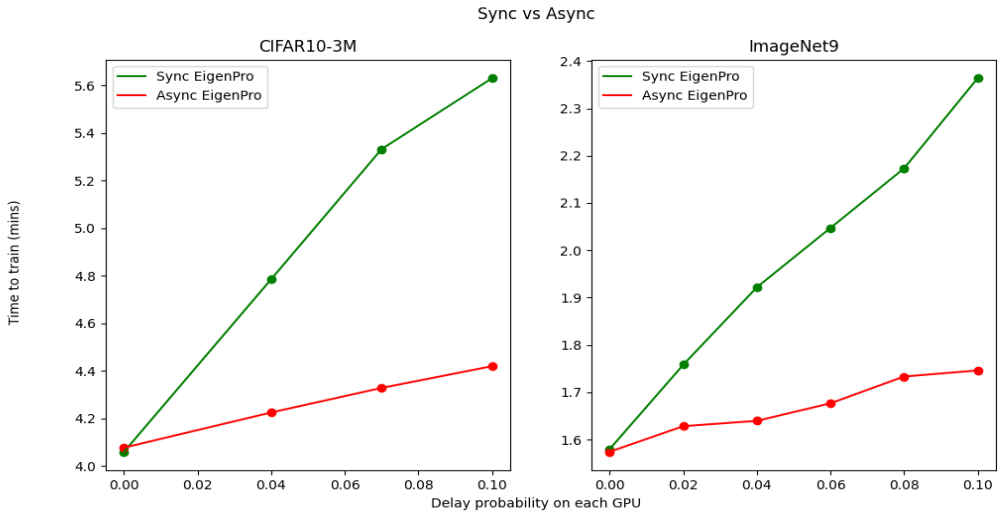

Figure 1: Comparison of synchronous against asynchronous EigenPro with 7 GPUs where each GPU can randomly delay gradient computation with probability given in x-axis. Time is for 1 epoch.

Table 3 shows speedup of asynchronous training without any delay model. For CIFAR10-3M and ImageNet9 asynchronous training we did not have to decrease the learning rate as compared to synchronous case. ... more discussion soon

Table 3: Speedup of asynchronous multi-GPU training. Time in minutes

	1 GPU		3 GPUs		7 GPUs	
	Time	Speedup	Time	Speedup	Time	Speedup
CIFAR10-3M	27.58	1x	9.47	**2.91**×	4.23	**6.52**×
ImageNet9	10.73	1x	3.71	**2.89**×	1.57	**6.83**×
HIGGS		1x				
TAXI		1x				

Table 4: Speedup of synchronous multi-GPU training. Time in minutes

	1 GPU		4 GPUs		8 GPUs	
	Time	Speedup	Time	Speedup	Time	Speedup
CIFAR10-3M	27.58	1x	7.04	**3.92**×	3.715	**7.42**×
ImageNet9	10.73	1x	2.74	**3.91**×	1.39	**7.71**×
ImageNet	8.241	1x	2.76	**2.98**×	1.30	**6.33**×
HIGGS	88.16	1x	22.20	**3.97**×	11.51	**7.65**×
TAXI	74.87	1x	18.80	**3.98**×	9.42	**7.95**×

References

[1] Yossi Arjevani, Ohad Shamir, and Nathan Srebro. A tight convergence analysis for stochastic gradient descent with delayed updates. In *Algorithmic Learning Theory*, pages 111–132. PMLR, 2020.

[2] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 07 2014.

[3] Léon Bottou. Online algorithms and stochastic approxima-p tions. *Online learning and neural networks*, 1998.

[4] Logan Engstrom, Andrew Ilyas, Hadi Salman, Shibani Santurkar, and Dimitris Tsipras. Robustness (python library), 2019.

[5] George S Kimeldorf and Grace Wahba. A correspondence between bayesian estimation on stochastic processes and smoothing by splines. *The Annals of Mathematical Statistics*, 41(2):495–502, 1970.

[6] Jian Li, Yong Liu, and Weiping Wang. Towards sharp analysis for distributed learning with random features. *arXiv preprint arXiv:1906.03155*, 2019.

[7] Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *International Conference on Machine Learning*, pages 3325–3334. PMLR, 2018.

[8] Siyuan Ma and Mikhail Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. *Advances in neural information processing systems*, 30, 2017.

[9] Siyuan Ma and Mikhail Belkin. Kernel machines that adapt to gpus for effective large batch training. *Proceedings of Machine Learning and Systems*, 1:360–373, 2019.

[10] Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I. Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 27(4):2202–2229, 2017.

[11] Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro Rudi. Kernel methods through the roof: handling billions of points efficiently. *Advances in Neural Information Processing Systems*, 33:14410–14422, 2020.

[12] Preetum Nakkiran, Behnam Neyshabur, and Hanie Sedghi. The deep bootstrap framework: Good online learners are good offline generalizers. In *International Conference on Learning Representations*, 2021.

[13] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, volume 24, 2011.

[14] L. F. Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 210:307–357, 1911.

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[16] Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.

[17] Yuchen Zhang, John Duchi, and Martin Wainwright. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *Journal of Machine Learning Research*, 16(102):3299–3340, 2015.

A Proofs

In many of the steps we will be using the following result

$$\mu_n \|\boldsymbol{v}\|_2^2 \leq \|\boldsymbol{M}\boldsymbol{v}\|_2^2 \leq \mu_1 \|\boldsymbol{v}\|_2^2 \tag{15}$$

where μ_1 is the largest eigenvalue and μ_n is the smallest eigenvalue of the full rank matrix $\boldsymbol{M}^\top \boldsymbol{M} \in \mathbb{R}^{n\times n}$. $\boldsymbol{v}$ is any vector.

Definition 2. Define preconditioner matrix $\boldsymbol{P}$ and reparametrization of $\boldsymbol{\alpha}$ as follows

$$\boldsymbol{P} := \boldsymbol{I} - \boldsymbol{M}\mathbf{K} \tag{16}$$

$$\boldsymbol{\beta} := \sqrt{\boldsymbol{P}}^{-1}\boldsymbol{\alpha} \tag{17}$$

Definition 3 (Selector matrix $\boldsymbol{Q}_m$)**.** For any index set $\mathcal{B}_m$ with size m we define a diagonal matrix which acts as a selector.

$$[\boldsymbol{Q}_m]_{ij} = \begin{cases} 1 & i = j, i \in \mathcal{B}_m \\ 0 & \text{otherwise} \end{cases}$$

Lemma 1 (Update equation for $\boldsymbol{\alpha}$)**.** *Updates in equation* (5) *and* (6) *can be merged into a single update equation as follows*

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t - \frac{\eta}{m}\boldsymbol{P}\boldsymbol{Q}_m\left(\mathbf{K}\boldsymbol{\alpha}_t - \boldsymbol{y}\right)$$

where $\boldsymbol{Q}_m$ is the selector matrix (definition 3) for the set of mini-batch indices

Proof. For analysis, we want to merge (5), (6) into a single update equation. We will use the selector matrix $\boldsymbol{Q}_m$ and $\boldsymbol{Q}_s$ for selecting set of mini-batch indices $\mathcal{B}_m$ and set of Nyström samples' indices $\mathcal{B}_s$ respectively. This gives a full update equation for $\boldsymbol{\alpha}$ with zeros in the non-batch and non-Nyström indices.

Now, merging (5) and (6) gives

$$\begin{aligned}\boldsymbol{\alpha}_{t+1} &= \boldsymbol{\alpha}_t - \frac{\eta}{m}\boldsymbol{Q}_m\left(\mathbf{K}\boldsymbol{\alpha}_t - \boldsymbol{y}\right) + \frac{\eta}{m}\boldsymbol{M}(\boldsymbol{Q}_m\mathbf{K}\boldsymbol{Q}_s)^\top\boldsymbol{Q}_m\left(\mathbf{K}\boldsymbol{\alpha}_t - \boldsymbol{y}\right) \\ &= \boldsymbol{\alpha}_t - \frac{\eta}{m}\left(\boldsymbol{I} - \boldsymbol{M}\mathbf{K}\right)\boldsymbol{Q}_m\left(\mathbf{K}\boldsymbol{\alpha}_t - \boldsymbol{y}\right)\end{aligned}$$

where, we have used $\boldsymbol{M}\boldsymbol{Q}_s = \boldsymbol{M},\ \boldsymbol{Q}_m^2 = \boldsymbol{Q}_m$. Define

$$\boldsymbol{P} := \boldsymbol{I} - \boldsymbol{M}\boldsymbol{K}$$

to get

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t - \frac{\eta}{m}\boldsymbol{P}\boldsymbol{Q}_m\left(\mathbf{K}\boldsymbol{\alpha}_t - \boldsymbol{y}\right)$$

Note that only $\mathcal{B}_m \cup \mathcal{B}_s$ indices are updated in each iteration just like in (5), (6). Now we have a single update equation for each iteration of EigenPro. □

Definition 4 (Selector $\boldsymbol{Q}_i^t$ for inconsistencies)**.** The updates from other processors might be partially complete when iteration t starts the update. To account for such scenarios define $\boldsymbol{Q}_i^t$. For iteration t and iteration i such that $t - \tau \leq i < t$, there exists a diagonal matrix $\boldsymbol{Q}_i^t \in \mathbb{R}^{n\times n}$ with diagonal entries in $\{0, 1\}$ such that

$$\widehat{\boldsymbol{\alpha}}_t - \boldsymbol{\alpha}_t = \frac{\eta}{m}\sum_{i=t-\tau}^{t-1}\boldsymbol{Q}_i^t\left\{\boldsymbol{P}\boldsymbol{Q}_m\left(\mathbf{K}\widehat{\boldsymbol{\alpha}}_i - \boldsymbol{y}\right)\right\} \tag{18}$$

i.e., between time of read and time of udpate, the difference in parameter is only because of delayed gradients and partial writes.

A.1 Proof of Theorem 1

Proof. First we will merge the two step udpate in equation (5), (6) into a single update equation in lemma 1. Then we derive a bound on $\|\boldsymbol{\alpha}_T - \boldsymbol{\alpha}^*\|^2_{\boldsymbol{P}^{-1}}$ for after T iteration. The upper bound involves two terms. The first term is an exponential convergence term since we are dealing with strong convexity and the second term is a variance term introduced by the delayed gradients.

Since for AsyncEigenPro, the updates are asynchronous, the parameter value on which $\boldsymbol{g}_m$ in equation (7) is computed could be different than the parameter value on which it updates.Also, there could be inconsistent reads which will result in noisy updates. We define $\widehat{\boldsymbol{\alpha}}$ to denote both delays (stale updates) and inconsistent reads. Using $\widehat{\boldsymbol{\alpha}}$ in the update rule of lemma 1 will give

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t - \frac{\eta}{m}\boldsymbol{P}\boldsymbol{Q}_m\left(\mathbf{K}\widehat{\boldsymbol{\alpha}}_t - \boldsymbol{y}\right)$$

We will perform the analysis of the iteration in a transformed space $\boldsymbol{\alpha} \mapsto \boldsymbol{\beta}$, where $\boldsymbol{\beta} := \sqrt{\boldsymbol{P}}^{-1}\boldsymbol{\alpha}$.

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \frac{\eta}{m}\sqrt{\boldsymbol{P}}\boldsymbol{Q}_m\left(\mathbf{K}\sqrt{\boldsymbol{P}}\widehat{\boldsymbol{\beta}_t} - \boldsymbol{y}\right)$$

We define a function $g_m : \mathbb{R}^n \to \mathbb{R}^n$

$$g_m(\boldsymbol{\beta}) := \frac{1}{m}\sqrt{\boldsymbol{P}}\boldsymbol{Q}_m\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right) \tag{19}$$

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \eta g_m(\widehat{\boldsymbol{\beta}_t}) \tag{20}$$

Note that, like in $\boldsymbol{\alpha}$, only $\mathcal{B}_m \cup \mathcal{B}_s$ indices are updated in each iteration of $\boldsymbol{\beta}$.
We see that

$$\|\boldsymbol{\beta} - \boldsymbol{\beta}^*\|_2^2 = \left\|\sqrt{\boldsymbol{P}}^{-1}\left(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*\right)\right\|_2^2 = \left(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*\right)^\top \boldsymbol{P}^{-1}\left(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*\right) = \|\boldsymbol{\alpha} - \boldsymbol{\alpha}^*\|^2_{\boldsymbol{P}^{-1}} \tag{21}$$

Using the above in Lemma 4 we get the theorem statement.

□

A.2 Results for $\boldsymbol{\beta}$

Definition 5. Define a function $h : \mathbb{R}^n \to \mathbb{R}$ as below

$$h(\boldsymbol{\beta}) := \frac{1}{n}\left(\frac{1}{2}\boldsymbol{\beta}^\top\sqrt{\boldsymbol{P}}\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{\beta}^\top\sqrt{\boldsymbol{P}}\boldsymbol{y}\right) \tag{22}$$

$$:= \frac{1}{n}\left(\frac{1}{2}\boldsymbol{\beta}^\top\mathbf{K}_p\boldsymbol{\beta} - \boldsymbol{\beta}^\top\mathbf{K}_p\boldsymbol{\beta}^*\right) \tag{23}$$

$$\text{where} \quad \mathbf{K}_p := \sqrt{\boldsymbol{P}}\mathbf{K}\sqrt{\boldsymbol{P}} \tag{24}$$

here, we have used $\boldsymbol{y} = \mathbf{K}\boldsymbol{\alpha}^* = \mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta}^*$

Lemma 2 (Eigensystems of matrices)**.** *Consider the Hessian matrix*

$$\nabla^2\mathcal{L}(\boldsymbol{w}) = \frac{1}{n}\boldsymbol{Z}^\top\boldsymbol{Z} := \boldsymbol{H} \tag{25}$$

and let $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n > 0$ be its eigenvalues corresponding to eigenvectors $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n$

(1) Top q eigenvalues of $\boldsymbol{Q}_s\boldsymbol{K}\boldsymbol{Q}_s$ are $s\lambda_1^{(s)}, s\lambda_2^{(s)}, \ldots, s\lambda_q^{(s)}$ with corresponding eigenvectors $\boldsymbol{e}_1^{(s)}, \boldsymbol{e}_2^{(s)}, \ldots, \boldsymbol{e}_q^{(s)}$

(2) Eigenvalues of $\boldsymbol{P} := \boldsymbol{I} - \boldsymbol{M}\mathbf{K}$ are

$$1 \geq 1 \geq \ldots \geq 1 \geq \frac{\lambda_{q+1}^{(s)}}{\lambda_q^{(s)}} \geq \ldots \geq \frac{\lambda_{q+1}^{(s)}}{\lambda_1^{(s)}}$$

and corresponding eigenvectors are $\boldsymbol{e}_n, \boldsymbol{e}_{n-1}, \ldots, \boldsymbol{e}_{q+1}, \boldsymbol{e}_q^{(s)}, \ldots, \boldsymbol{e}_1^{(s)}$

(3) Eigenvalues of $\mathbf{K}_p := \sqrt{\boldsymbol{P}}\mathbf{K}\sqrt{\boldsymbol{P}}$ *are*

$$s\lambda_{q+1}^{(s)} \geq s\lambda_{q+1}^{(s)} \geq \ldots \geq s\lambda_{q+1}^{(s)} \geq n\lambda_{q+1} \geq \ldots \geq n\lambda_n$$

and corresponding eigenvectors are $\boldsymbol{e}_1^{(s)}, \boldsymbol{e}_2^{(s)}, \ldots, \boldsymbol{e}_q^{(s)}, \boldsymbol{e}_{q+1}, \ldots, \boldsymbol{e}_n$

Proof.

(1) $s\lambda_1^{(s)}, s\lambda_2^{(s)}, \ldots, s\lambda_q^{(s)}$ are eigenvalues of $\boldsymbol{K}_{ss} \in \mathbb{R}^{s\times s}$ by definition. We can construct $\boldsymbol{e}_i^{(s)}$ from eigenvectors of $\boldsymbol{K}_{ss}$ by copying s dimensions of the i^{th} eigenvector of $\boldsymbol{K}_{ss}$ into the corresponding dimensions of $\boldsymbol{e}_i^{(s)}$ and the rest of the $n-s$ dimensions of $\boldsymbol{e}_i^{(s)}$ can be set to 0. Then

$$\boldsymbol{Q}_s\boldsymbol{K}\boldsymbol{Q}_s\boldsymbol{e}_i^{(s)} = s\lambda_i^{(s)}\boldsymbol{e}_i^{(s)} \quad \forall i = 1, 2, \ldots, s$$

Rest of the $n-s$ eigenvectors can be standard basis vectors with 0 eigenvalue. Finally, $\left\langle \boldsymbol{e}_i^{(s)}, \boldsymbol{e}_j^{(s)} \right\rangle = 0 \quad \forall i \neq j$

(2) By definition of $\boldsymbol{M}$, we can write $\boldsymbol{M} = \boldsymbol{Q}_s\boldsymbol{M} = \boldsymbol{M}\boldsymbol{Q}_s = \boldsymbol{Q}_s\boldsymbol{M}\boldsymbol{Q}_s$. Similarly, $\boldsymbol{e}_i^{(s)} = \boldsymbol{Q}_s\boldsymbol{e}_i^{(s)}$. Inner product of Nyström approximated eigenvector and exact eigenvector is 0.

$$\left\langle \boldsymbol{e}_i^{(s)}, \boldsymbol{e}_j \right\rangle = \frac{1}{\lambda_j}\left\langle \boldsymbol{e}_i^{(s)}, \boldsymbol{K}\boldsymbol{e}_j \right\rangle = \frac{1}{n\lambda_j}\left\langle \boldsymbol{K}\boldsymbol{Q}_s\boldsymbol{e}_i^{(s)}, \boldsymbol{e}_j \right\rangle \overset{(a)}{=} \frac{1}{n\lambda_j c}\left\langle \boldsymbol{e}_i, \boldsymbol{e}_j \right\rangle = 0 \tag{26}$$

where (a) follows from Nyström approximation

$$\boldsymbol{e}_i = c\boldsymbol{K}\boldsymbol{Q}_s\boldsymbol{e}_i^{(s)} = c\boldsymbol{K}\boldsymbol{e}_i^{(s)} \tag{27}$$

where c is a constant scaling factor which depends on $s, n, \lambda_i^{(s)}$.
$\boldsymbol{M}$ can be expressed in terms of eigenvectors as follows

$$\boldsymbol{M} = \sum_{i=1}^{q}\left(1 - \frac{\lambda_{q+1}^{(s)}}{\lambda_i^{(s)}}\right)\frac{1}{s\lambda_i^{(s)}}\boldsymbol{e}_i^{(s)}\boldsymbol{e}_i^{(s)\top} \tag{28}$$

Next, we verify the eigenvectors

$$\begin{aligned}\boldsymbol{P}\boldsymbol{e}_i^{(s)} &= \boldsymbol{e}_i^{(s)} - \boldsymbol{M}\boldsymbol{K}\boldsymbol{e}_i^{(s)} = \boldsymbol{e}_i^{(s)} - \boldsymbol{M}\boldsymbol{Q}_s\boldsymbol{K}\boldsymbol{Q}_s\boldsymbol{e}_i^{(s)} = \boldsymbol{e}_i^{(s)} - \boldsymbol{M}s\lambda_i^{(s)}\boldsymbol{e}_i^{(s)} \\ &\overset{(a)}{=} \boldsymbol{e}_i^{(s)} - \left(1 - \frac{\lambda_{q+1}^{(s)}}{\lambda_i^{(s)}}\right)\boldsymbol{e}_i^{(s)} = \frac{\lambda_{q+1}^{(s)}}{\lambda_i^{(s)}}\boldsymbol{e}_i^{(s)} \qquad \forall i = 1, 2, \ldots, q\end{aligned}$$

where (a) follows from (28)

$$\begin{aligned}\boldsymbol{P}\boldsymbol{e}_j &= \boldsymbol{e}_j - \boldsymbol{M}\boldsymbol{K}\boldsymbol{e}_j = \boldsymbol{e}_j - n\lambda_j\boldsymbol{M}\boldsymbol{e}_j \\ &\overset{(a)}{=} \boldsymbol{e}_j \qquad \forall j = q+1, \ldots, n\end{aligned}$$

where (a) follows from (28) and (26)

(3) Verifying eigenvectors

$$\begin{aligned}
\boldsymbol{K}_p\boldsymbol{e}_i^{(s)} = \sqrt{\boldsymbol{P}}\boldsymbol{K}\sqrt{\boldsymbol{P}}\boldsymbol{e}_i^{(s)} &= \sqrt{\frac{\lambda_{q+1}^{(s)}}{\lambda_i^{(s)}}}\sqrt{\boldsymbol{P}}\boldsymbol{K}\boldsymbol{e}_i^{(s)} \\
&\overset{(a)}{=} \frac{1}{c}\sqrt{\frac{\lambda_{q+1}^{(s)}}{\lambda_i^{(s)}}}\sqrt{\boldsymbol{P}}\boldsymbol{e}_i \\
&\overset{(b)}{=} \frac{1}{c}\frac{\lambda_{q+1}^{(s)}}{\lambda_i^{(s)}}\left\langle \boldsymbol{e}_i^{(s)}, \boldsymbol{e}_i \right\rangle \boldsymbol{e}_i^{(s)} \\
&\overset{(c)}{=} \frac{\lambda_{q+1}^{(s)}}{\lambda_i^{(s)}}\left\langle \boldsymbol{e}_i^{(s)}, \boldsymbol{K}\boldsymbol{e}_i^{(s)} \right\rangle \boldsymbol{e}_i^{(s)} \\
&= \frac{\lambda_{q+1}^{(s)}}{\lambda_i^{(s)}}\left\langle \boldsymbol{e}_i^{(s)}, \boldsymbol{Q}_s\boldsymbol{K}\boldsymbol{Q}_s\boldsymbol{e}_i^{(s)} \right\rangle \boldsymbol{e}_i^{(s)} \\
&\overset{(e)}{=} \frac{\lambda_{q+1}^{(s)}}{\lambda_i^{(s)}} s\lambda_i^{(s)}\left\langle \boldsymbol{e}_i^{(s)}, \boldsymbol{e}_i^{(s)} \right\rangle \boldsymbol{e}_i^{(s)} \\
&= s\lambda_{q+1}^{(s)}\boldsymbol{e}_i^{(s)} \qquad i = 1, 2, \ldots, q
\end{aligned}$$

where (a), (c) follows from(27), (b) from eigensystem of $\boldsymbol{P}$ and (26), (e) follows from eigensystem of $\boldsymbol{Q}_s\boldsymbol{K}\boldsymbol{Q}_s$

$$\boldsymbol{K}_p\boldsymbol{e}_j = \sqrt{\boldsymbol{P}}\boldsymbol{K}\sqrt{\boldsymbol{P}}\boldsymbol{e}_j = \sqrt{\boldsymbol{P}}\boldsymbol{K}\boldsymbol{e}_j = \sqrt{\boldsymbol{P}}n\lambda_j\boldsymbol{e}_j = n\lambda_j\boldsymbol{e}_j \qquad \forall j = q+1, \ldots, n$$

□

Lemma 3. *Properties of $g_m(\boldsymbol{\beta})$ defined in* (19)

(1) Unbiased estimator

$$\mathbb{E}_m\left[g_m(\boldsymbol{\beta})\right] = \nabla h(\boldsymbol{\beta}) \tag{29}$$

(2) Decomposing $g_m(\boldsymbol{\beta})$

$$\mathbb{E}_m \left\|g_m(\boldsymbol{\beta})\right\|_2^2 = \frac{1}{m}\mathbb{E}_{i_1}\left\|g_{1,i_1}(\boldsymbol{\beta})\right\|_2^2 + \frac{m-1}{m}\left\|\nabla h(\boldsymbol{\beta})\right\|_2^2 \tag{30}$$

$$\text{where} \quad g_{1,i_k}(\boldsymbol{\beta}) := \sqrt{\boldsymbol{P}}\boldsymbol{Q}_{i_k}\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right) \tag{31}$$

(3) Bound on $g_m(\boldsymbol{\beta})$

$$\mathbb{E}\left\|g_m(\boldsymbol{\beta})\right\|_2^2 \leq \frac{\mathcal{L}_0}{m} \tag{32}$$

where $\mathcal{L}_0$ is the value of loss (1) *at initialization.*

Proof.

(1)

$$\begin{aligned}
\mathbb{E}_m\left[g_m(\boldsymbol{\beta})\right] &= \frac{1}{m}\sqrt{\boldsymbol{P}}\mathbb{E}_m[\boldsymbol{Q}_m]\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right) \\
&= \frac{1}{m}\sqrt{\boldsymbol{P}}\left(\frac{m}{n}\boldsymbol{I}\right)\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right) \\
&= \frac{1}{n}\sqrt{\boldsymbol{P}}\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right) = \nabla h(\boldsymbol{\beta})
\end{aligned}$$

(2) Let the set of i.i.d. indices for the mini-batch be $\boldsymbol{R}_m = \{i_k : k \leq m, k \in \mathbb{N}\}$. Then, g_m can be written as sum of terms where each term deals with an independent random variable.

$$g_m(\boldsymbol{\beta}) = \frac{1}{m}\sum_{k=1}^{m} \sqrt{\boldsymbol{P}}\boldsymbol{Q}_{i_k}\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right)$$

$$g_m(\boldsymbol{\beta}) = \frac{1}{m}\sum_{k=1}^{m} g_{1,i_k}(\boldsymbol{\beta})$$

$$\begin{aligned}
\mathbb{E}_m \left\|g_m(\boldsymbol{\beta})\right\|_2^2 &= \mathbb{E}_m \left\langle \frac{1}{m}\sum_{k=1}^{m} g_{1,i_k}(\boldsymbol{\beta}),\ \frac{1}{m}\sum_{k=1}^{m} g_{1,i_k}(\boldsymbol{\beta}) \right\rangle \\
&= \frac{1}{m^2}\sum_{k=1}^{m} \mathbb{E}_{i_k} \left\|g_{1,i_k}(\boldsymbol{\beta})\right\|_2^2 + \frac{1}{m^2}\sum_{\substack{j,k \\ j\neq k}} \mathbb{E}_{i_k,i_j} \left\langle g_{1,i_k}(\boldsymbol{\beta}),\ g_{1,i_j}(\boldsymbol{\beta})\right\rangle \\
&= \frac{1}{m}\mathbb{E}_{i_1} \left\|g_{1,i_1}(\boldsymbol{\beta})\right\|_2^2 + \frac{m-1}{m}\left\|\nabla h(\boldsymbol{\beta})\right\|_2^2
\end{aligned}$$

(3)

$$\begin{aligned}
\mathbb{E}\left\|g_m(\boldsymbol{\beta})\right\|_2^2 &= \frac{1}{m^2}\mathbb{E}\left\|\sqrt{\boldsymbol{P}}\boldsymbol{Q}_m\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right)\right\|_2^2 \\
&\leq \frac{1}{m^2}\mathbb{E}\left\|\boldsymbol{Q}_m\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right)\right\|_2^2 \\
&= \frac{1}{m^2}\mathbb{E}\left\langle \mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}, \mathbb{E}_m[\boldsymbol{Q}_m]\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right)\right\rangle \\
&= \frac{1}{mn}\mathbb{E}\left\|\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right)\right\|_2^2 \\
&\overset{(a)}{\leq} \frac{1}{mn}\left\|\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta}_0 - \boldsymbol{y}\right)\right\|_2^2 \\
&= \frac{\mathcal{L}_0}{m}
\end{aligned}$$

In (a), we assume that the expected value of loss is less than initial loss. where $\mathcal{L}_0$ is the value of loss (1) at initialization.

□

Lemma 4 (Convergence of $\boldsymbol{\beta}$)**.** *If $\eta \leq \frac{m}{\nu + \lambda_{q+1}^{(s)}(m-1)}$, then the iteration given in* (20) *satisfies the following after T iterations*

$$\mathbb{E}\left\|\boldsymbol{\beta}_T - \boldsymbol{\beta}^*\right\|_2^2 \leq \left(1 - \frac{\eta\lambda_n}{2}\right)^T \left\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}^*\right\|_2^2 + \frac{2\eta^2\tau\mathcal{L}_0}{m}\left(1 - \left(1 - \frac{\eta\lambda_n}{2}\right)^T\right) \tag{33}$$

Here, $\nu = max_i[\mathbf{K}]_{ii}$, τ is the maximum delay in gradients, $\mathcal{L}_0$ is the loss (1) *at initialization.*

Proof. $h(\boldsymbol{\beta})$ is $\lambda_{q+1}^{(s)}$-smooth and λ_n-strongly convex (See definition 5 and Lemma 2).

From (4) $\boldsymbol{y} = \mathbf{K}\boldsymbol{\alpha}^* = \mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta}*$ which implies $h(\boldsymbol{\beta}^*) = 0$

$$h(\boldsymbol{\beta}) - h(\boldsymbol{\beta}^*) = h(\boldsymbol{\beta}) \leq \langle \nabla h(\boldsymbol{\beta}), \boldsymbol{\beta} - \boldsymbol{\beta}^*\rangle + \frac{\lambda_n}{2}\left\|\boldsymbol{\beta} - \boldsymbol{\beta}^*\right\|_2^2$$

Analysing the norm $\|\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}^*\|_2^2$

$$
\begin{aligned}
&\mathbb{E}\,\|\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}^*\|_2^2 \\
&= \mathbb{E}\left\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^* - \eta g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\|_2^2 \\
&= \mathbb{E}\,\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2^2 - 2\eta\mathbb{E}\left\langle \boldsymbol{\beta}_t - \boldsymbol{\beta}^*,\ g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\rangle + \eta^2\mathbb{E}\left\|g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\|_2^2 \\
&= \mathbb{E}\,\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2^2 - 2\eta\mathbb{E}\left\langle \widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}^*,\ g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\rangle - 2\eta\mathbb{E}\left\langle \widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}_t,\ g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\rangle + \eta^2\mathbb{E}\left\|g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\|_2^2 \\
&\overset{(a)}{=} \mathbb{E}\,\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2^2 - 2\eta\mathbb{E}\left\langle \widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}^*,\ g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\rangle + \eta^2\mathbb{E}\left\|g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\|_2^2 \\
&\overset{(b)}{=} \mathbb{E}\,\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2^2 - 2\eta\mathbb{E}\left\langle \widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}^*,\ \nabla h(\widehat{\boldsymbol{\beta}}_t)\right\rangle + \eta^2\mathbb{E}\left\|g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\|_2^2 \\
&\overset{(c)}{\leq} \mathbb{E}\,\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2^2 - 2\eta\left(\mathbb{E}\left[h(\widehat{\boldsymbol{\beta}}_t)\right] + \frac{\lambda_n}{2}\mathbb{E}\left\|\widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}^*\right\|_2^2\right) + \eta^2\mathbb{E}\left\|g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\|_2^2 \\
&\overset{(d)}{\leq} \mathbb{E}\,\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2^2 - 2\eta\left(\mathbb{E}\left[h(\widehat{\boldsymbol{\beta}}_t)\right] + \frac{\lambda_n}{4}\mathbb{E}\,\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2^2 - \frac{\lambda_n}{2}\mathbb{E}\left\|\widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}_t\right\|_2^2\right) + \eta^2\mathbb{E}\left\|g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\|_2^2 \\
&= \left(1 - \frac{\eta\lambda_n}{2}\right)\mathbb{E}\,\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2^2 + \eta\lambda_n\mathbb{E}\left\|\widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}_t\right\|_2^2 - 2\eta\left(\mathbb{E}\left[h(\widehat{\boldsymbol{\beta}}_t)\right] - \frac{\eta}{2}\mathbb{E}\left\|g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\|_2^2\right) \\
&\overset{(e)}{\leq} \left(1 - \frac{\eta\lambda_n}{2}\right)\mathbb{E}\,\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2^2 + \frac{\eta^3\tau\lambda_n}{m}\mathcal{L}_0
\end{aligned}
$$

where (a) is from Lemma 5, (b) follows from (29), (c) is due to strong convexity of h, (d) uses triangle inequality of norms and (e) follows from Lemma 7 , Lemma 6

Finally,

$$
\begin{aligned}
\mathbb{E}\,\|\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}^*\|_2^2 &\leq \left(1 - \frac{\eta\lambda_n}{2}\right)^2\|\boldsymbol{\beta}_{t-1} - \boldsymbol{\beta}^*\|_2^2 + \frac{\eta^3\tau\lambda_n\mathcal{L}_0}{m}\left(1 + \left(1 - \frac{\eta\lambda_n}{2}\right)\right) \\
\mathbb{E}\,\|\boldsymbol{\beta}_T - \boldsymbol{\beta}^*\|_2^2 &\leq \left(1 - \frac{\eta\lambda_n}{2}\right)^T\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}^*\|_2^2 + \frac{2\eta^2\tau\mathcal{L}_0}{m}\left(1 - \left(1 - \frac{\eta\lambda_n}{2}\right)^T\right)
\end{aligned}
$$

□

Lemma 5 (Non overlapping updates)**.** *All the changes to the parameter vector between the time a processor reads the parameter and updates it is in the orthogonal direction of the update to be done.*

$$\left\langle \widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}_t,\ g_{m_t}(\widehat{\boldsymbol{\beta}}_t)\right\rangle = 0 \tag{34}$$

Proof. Let r be the processor index that is updating at time t. Recall from Section 3 that each processor r updates indices of $\mathcal{A}lphavec$ belonging to data $\mathcal{D}_r$ exclusively. Let the index set of this data $\mathcal{D}_r$ be $\mathcal{A}_r$. Each processor r has different preconditioner $\boldsymbol{P}_r$ whose Nystöm indices are chosen from $\mathcal{A}_r$ which results in $\boldsymbol{P}_r$ transforming only a subset $\mathcal{A}_r$ indices i.e.,

$$[\boldsymbol{P}_r]_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \qquad \forall i \notin \mathcal{A}_r \tag{35}$$

So, only a subset of $\mathcal{A}_r$ indices are updated in $\boldsymbol{\beta} = \sqrt{\boldsymbol{P}_r}\boldsymbol{\alpha}$ too. Also note only processor r can update indices in $\mathcal{A}_r$ and updates from one processor is sequential. So, between the time of read and time of update, the value in parameter vector corresponding to indices in $\mathcal{A}_r$ has not changed i.e.,

$$\left[\widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}_t\right]_i = 0 \qquad \forall i \in \mathcal{A}_r$$

Also by definition of g_m

$$[g_{m_t}(\widehat{\boldsymbol{\beta}}_t)]_i = 0 \qquad\qquad \forall i \notin \mathcal{A}_r$$

which leads to

$$\left\langle \widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}_t,\ g_{m_t}(\widehat{\boldsymbol{\beta}}_t) \right\rangle = 0$$

□

Lemma 6 (Bounding the perturbation)**.** *At all times the norm of the change in parameter between the time of read and time of update is bounded as follows*

$$\mathbb{E}\left\|\widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}_t\right\|_2^2 \leq \frac{\eta^2 \tau \mathcal{L}_0}{m} \tag{36}$$

where τ is the maximum possible delay in gradients and $\mathcal{L}_0$ is the value of loss (1) *at initialization.*

Proof. Similar to (18) the change of parameter vector between the time of read and time of update can be given as

$$\widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta} = \eta \sum_{i=t-\tau}^{t-1} \boldsymbol{Q}_i^t g_{m_i}(\widehat{\boldsymbol{\beta}}_i)$$

Then,

$$\begin{aligned}
\mathbb{E}\left\|\widehat{\boldsymbol{\beta}}_t - \boldsymbol{\beta}_t\right\|_2^2 &= \mathbb{E}\left\langle \sum_{i=t-\tau}^{t-1} \eta \boldsymbol{Q}_i^t g_{m_i}(\widehat{\boldsymbol{\beta}}_i),\ \sum_{i=t-\tau}^{t-1} \eta \boldsymbol{Q}_i^t g_{m_i}(\widehat{\boldsymbol{\beta}}_i) \right\rangle \\
&= \eta^2 \sum_{i=t-\tau}^{t-1} \mathbb{E}\left\|\boldsymbol{Q}_i^t g_{m_i}(\widehat{\boldsymbol{\beta}})\right\|_2^2 + \eta^2 \sum_{\substack{i,j \\ i\neq j}} \mathbb{E}\left\langle \boldsymbol{Q}_i^t g_{m_i}(\widehat{\boldsymbol{\beta}}_i), \boldsymbol{Q}_j^t g_{m_j}(\widehat{\boldsymbol{\beta}}_j) \right\rangle \\
&\overset{(a)}{=} \eta^2 \sum_{i=t-\tau}^{t-1} \mathbb{E}\left\|\boldsymbol{Q}_i^t g_{m_i}(\widehat{\boldsymbol{\beta}}_i)\right\|_2^2 \\
&\overset{(b)}{\leq} \eta^2 \sum_{i=t-\tau}^{t-1} \mathbb{E}\left\|g_{m_i}(\widehat{\boldsymbol{\beta}}_i)\right\|_2^2 \\
&\overset{(c)}{\leq} \frac{\eta^2 \tau \mathcal{L}_0}{m}
\end{aligned}$$

Recall from Section 3 that updates from each processor is non-overlapping. Here g_{m_i}, g_{m_j} are updates from different processors. (a) follow from this fact. (b) is from the definition of $\boldsymbol{Q}_i^t$ which is a diagonal matrix with 0 or 1 in the diagonal entries. (c) is from lemma 3. □

Lemma 7 (Interpolation to variance reduction)**.** *For $h(\boldsymbol{\beta})$ defined in definition 5, we show the following*

$$\mathbb{E}_m\left[h(\boldsymbol{\beta}) - \frac{\eta}{2}\left\|g_m(\boldsymbol{\beta})\right\|_2^2\right] \geq 0 \tag{37}$$

$$\text{if} \quad \eta \leq \frac{m}{\nu + \lambda_{q+1}^{(s)}(m-1)} \tag{38}$$

where $\nu = max_i[\mathbf{K}]_{ii}$

Proof. Define a function $\bar{h}(\boldsymbol{\alpha}) : \mathbb{R}^n \to \mathbb{R}$ as

$$\bar{h} := \frac{1}{n}\left(\boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} - \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha}^*\right)$$

For an arbitrary index i_k, define a function $\bar{H}_{i_k} : \mathbb{R}^n \to \mathbb{R}$ as follows

$$\bar{H}_{i_k}(\boldsymbol{\beta}) := \mathbf{1}^\top Q_{i_k}[\bar{h}_1(\alpha_1), \bar{h}_2(\alpha_2), \ldots, \bar{h}_n(\alpha_n)]^\top \tag{39}$$

$$\bar{h}_i(\alpha_i) := \bar{h}_i(\alpha_i; \alpha_{\setminus i}) := \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha} - \boldsymbol{\alpha}^\top \mathbf{K}\boldsymbol{\alpha}^* \tag{40}$$

$\bar{h}_i(\alpha_i) : \mathbb{R} \to \mathbb{R}$ treats the i^{th} dimension of $\boldsymbol{\alpha}$ as variable and rest of the dimensions of $\boldsymbol{\alpha}$ as constants. Note that for $\boldsymbol{\alpha} = \sqrt{\boldsymbol{P}}\boldsymbol{\beta}$

$$\mathbb{E}_{i_k}[\bar{H}_{i_k}(\boldsymbol{\alpha})] = \bar{h}(\boldsymbol{\alpha}) = h(\boldsymbol{\beta}) \tag{41}$$

$$\nabla \bar{H}_{i_k}(\boldsymbol{\alpha}) = \boldsymbol{Q}_{i_k}\mathbf{K}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) \tag{42}$$

$$\nabla^2 \bar{H}_{i_k}(\boldsymbol{\alpha}) = \boldsymbol{Q}_{i_k}\mathbf{K}\boldsymbol{Q}_{i_k} \tag{43}$$

Define $\nu := max_i\ [\mathbf{K}]_{ii}$. We see that $\forall_i \bar{h}_i(\alpha_i)$ is ν-smooth and also $\bar{H}_{i_k}(\boldsymbol{\alpha})$ is ν-smooth

$$\bar{H}_{i_k}(\boldsymbol{\alpha}) - \frac{1}{2\nu}\left\|\nabla \bar{H}_{i_k}(\boldsymbol{\alpha})\right\|_2^2 \geq 0 \tag{44}$$

Relating g_{1,i_k} defined in lemma 3 to $\nabla \bar{H}_{i_k}$

$$\mathbb{E}_{i_k}\left\|g_{1,i_k}(\boldsymbol{\beta})\right\|_2^2 = \mathbb{E}_{i_k}\left\|\sqrt{\boldsymbol{P}}\boldsymbol{Q}_{i_k}\left(\mathbf{K}\sqrt{\boldsymbol{P}}\boldsymbol{\beta} - \boldsymbol{y}\right)\right\|_2^2 \tag{45}$$

$$= \mathbb{E}_{i_k}\left\|\sqrt{\boldsymbol{P}}\boldsymbol{Q}_{i_k}\mathbf{K}\sqrt{\boldsymbol{P}}(\boldsymbol{\beta} - \boldsymbol{\beta}^*)\right\|_2^2 \tag{46}$$

$$\overset{(a)}{\leq} \mathbb{E}_{i_k}\left\|\boldsymbol{Q}_{i_k}\mathbf{K}\sqrt{\boldsymbol{P}}(\boldsymbol{\beta} - \boldsymbol{\beta}^*)\right\|_2^2 \tag{47}$$

$$= \mathbb{E}_{i_k}\left\|\boldsymbol{Q}_{i_k}\mathbf{K}(\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)\right\|_2^2 = \mathbb{E}_{i_k}\left\|\nabla \bar{H}_{i_k}(\boldsymbol{\alpha})\right\|_2^2 \tag{48}$$

where (a) uses lemma 2

Using (30)

$$\mathbb{E}_m\left[h(\boldsymbol{\beta}) - \frac{\eta}{2}\left\|g_{m_t}(\boldsymbol{\beta})\right\|_2^2\right] \geq h(\boldsymbol{\beta}) - \frac{\eta}{2m}\mathbb{E}_{i_1}\left\|g_{1,i_1}(\boldsymbol{\beta})\right\|_2^2 - \frac{\eta(m-1)}{2m}\left\|\nabla h(\boldsymbol{\beta})\right\|_2^2$$

Choosing appropriate $\eta(p)$ for $p \in [0,1]$ and using (41)

$$\begin{aligned}\mathbb{E}_m\left[h(\boldsymbol{\beta}) - \frac{\eta}{2}\left\|g_m(\boldsymbol{\beta})\right\|_2^2\right] &\geq \mathbb{E}_{i_1}\left[p\bar{H}_{i_1}(\boldsymbol{\alpha}) - \frac{\eta(p)}{2m}\left\|g_{1,i_1}(\boldsymbol{\beta})\right\|_2^2\right] + (1-p)h(\boldsymbol{\beta}) - \frac{\eta(p)(m-1)}{2m}\left\|\nabla h(\boldsymbol{\beta})\right\|_2^2 \\ &\overset{(a)}{\geq} \mathbb{E}_{i_1}\left[p\bar{H}_{i_1}(\boldsymbol{\alpha}) - \frac{\eta(p)}{2m}\left\|\nabla \bar{H}_{i_1}(\boldsymbol{\alpha})\right\|_2^2\right] + (1-p)h(\boldsymbol{\beta}) - \frac{\eta(p)(m-1)}{2m}\left\|\nabla h(\boldsymbol{\beta})\right\|_2^2\end{aligned}$$

where (a) is from (48)

Solving $\eta(p) \leq \min\left\{\frac{mp}{\nu}, \frac{m(1-p)}{\lambda_{q+1}^{(s)}(m-1)}\right\}$ for $p \in [0,1]$ we get $\eta \leq \frac{m}{\nu + \lambda_{q+1}^{(s)}(m-1)}$. Now we can use smoothness below

$$\begin{aligned}\mathbb{E}_m\left[h(\boldsymbol{\beta}) - \frac{\eta}{2}\left\|g_m(\boldsymbol{\beta})\right\|_2^2\right] &\geq p\mathbb{E}_{i_1}\left[\bar{H}_{i_1}(\boldsymbol{\alpha}) - \frac{1}{2\nu}\left\|\nabla \bar{H}_{i_1}(\boldsymbol{\alpha})\right\|_2^2\right] + (1-p)\left[h(\boldsymbol{\beta}) - \frac{1}{2\lambda_{q+1}^{(s)}}\left\|\nabla h(\boldsymbol{\beta})\right\|_2^2\right] \\ &\geq 0\end{aligned}$$

□