Randomized Sketching for Large-Scale Sparse Ridge Regression Problems

Chander Iyer*, Christopher Carothers*, and Petros Drineas†
*Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, USA
†Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA

***Abstract*—We present a fast randomized ridge regression solver for sparse overdetermined matrices in distributed-memory platforms. Our solver is based on the Blendenpik algorithm, but employs sparse random projection schemes to construct a sketch of the input matrix. These sparse random projection sketching schemes, and in particular the use of the Randomized Sparsity-Preserving Transform, enable our algorithm to scale the distributed memory vanilla implementation of Blendenpik and provide up to $\times 13$ speedup over a state-of-the-art parallel Cholesky-like sparse-direct solver.**

***Index Terms*—Linear algebra, high performance computing, least-squares approximation, sparse matrices**

I. Introduction

Least-squares regression is one of the most widely used routines in statistical data analysis for a variety of application domains. The least-squares regression problem is defined as follows: given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{R}^m$, we seek to compute

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 . \tag{1}$$

We are particularly interested in regularized least-squares regression or ridge regression problems, where a loss function, commonly referred to as regularization, is imposed in order to deal with highly ill-conditioned input matrices $\mathbf{A}$. This standard setting is defined as follows: given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, a vector $\mathbf{b} \in \mathbb{R}^m$ and a regularization parameter λ, we seek to compute the minimum length solution

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_2^2 . \tag{2}$$

Equivalently for matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ that are overdetermined $(m > n)$, this can be reduced to the following problem:

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathbb{R}^n} \left\| \begin{pmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{pmatrix} \mathbf{x} - \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix} \right\|_2^2 . \tag{3}$$

Several direct and iterative algorithms have been proposed to solve ridge regression efficiently for sparse and large overdetermined systems [1], returning solutions whose accuracy is close to machine precision. While current state-of-the-art approaches solve the ridge regression problem in the dual space [2] or use kernelized ridge regression [3], scalable implementations still run in $O(mn^2)$ time (assuming $m > n$). The focus of our work is the design and implementation of a scalable ridge regression solver for sparse matrices.

This work is partially supported by NSF IIS-1302231.

Recent years have witnessed an explosion of research on so-called Randomized Numerical Linear Algebra (or RandNLA for short) algorithms, which leverage the power of randomization in order to perform standard matrix computations. One of the core problems that have been extensively researched in this emerging field is the least-squares regression problem of eqn. (1). Sarlos [4] and Drineas *et al.* [5] introduced the first randomized algorithms for this problem. These algorithms are based on the application of the sub-sampled Randomized Hadamard Transform(SRHT) to columns of the input matrix in order to create a least-squares problem of smaller size that can be then solved exactly and whose solution provably approximates the solution of the original problem with very high probability. This was followed by the work of Rokhlin and Tygert [6], who used a subsampled Randomized Fourier Transform(SRFT) to form a preconditioner and then used a standard iterative solver to solve the preconditioned problem. At the same time, Avron *et al.* [7] introduced Blendenpik, an algorithm and a software package which was the first practical implementation of a RandNLA dense least-squares solver that consistently and comprehensively outperformed state-of-the-art implementations of the traditional QR-based $O(mn^2)$ algorithms. Since then, there has been extensive research on RandNLA algorithms for regression problems; see Yang *et al.* [8] for a recent survey.

One of the earliest approaches to solve the ridge regression problem of eqn. (2) was proposed by [9] using SRHT to accelerate the computation of the kernel matrix $\mathbf{A}\mathbf{A}^T$ in the dual space. Their algorithm runs in $O(mn\log(m)/\sqrt{\epsilon} + m^3)$ time for underdetermined systems($m \ll n$). This was subsequently improved by [10] to run in $O(\text{nnz}(\mathbf{A}) + m^3/\epsilon^2)$ time by incorporating a combination of a sparse embedding transform proposed by Clarkson and Woodruff [11] and the SRHT to compute a sketch of the matrix, which is subsequently used to solve the primal problem of eqn. 2. However, both of these approaches focus on the single-processor setting; in fact, most research on randomized ridge regression algorithms has focused on the single processor setting, with an important exception. Meng *et al.* [12] introduced LSRN, a distributed memory algorithm for regularized least-squares problems based on random Gaussian projections. While the algorithm is still an $O(mn^2)$ algorithm, the benefits of randomization are apparent with respect to both constants in the asymptotic analysis, as well as its much improved efficiency on parallel environments.

We explore the behavior of Blendenpik-type algorithms in a distributed memory setting for sparse matrices in this work.

We show that our implementation of the sparse embedding transform proposed by Clarkson and Woodruff (henceforth referred to as Randomized Sparsity-Preserving Transform or RSPT for short) and a combination of RSPT and the Randomized Discrete Cosine Transform (RDCT) lead to speedups that are not only faster than state-of-the-art distributed baseline solvers but are also able to scale to much larger matrix dimensions, while providing near-optimal solutions. Our implementation and experiments were run on AMOS[1], the high-performance Blue Gene/Q supercomputer system at Rensselaer. AMOS has five racks, 5,120 nodes (81,920 cores), and 81,920 GB of main memory, a peak performance of one PetaFLOP (10^{15} floating point operations per second), and a 5-D torus network with 2 GB/sec of bandwidth per link and 512 GB/sec to 1 TB/sec of bisection network bandwidth per rack (depending on the torus network configuration). Due to runtime constraints imposed by the scheduling system for each partition of AMOS, we limited our experiments to partitions containing 128 nodes (2048 cores). The Blue Gene/Q architecture supports a hybrid communication framework that uses the MPI (Message Passing Interface) [13] standard for distributed communication and multithreading using OpenMP [14].

Our main contributions in this paper are[2]: **(i)** the implementation of the sparse embedding RSPT and a combination of RSPT and a batchwise implementation scheme of the RDCT in the context of the Blendenpik algorithm on distributed-memory platforms; and **(ii)** a detailed evaluation of the sparse randomized sketching transforms in the context of the Blendenpik algorithm and their parameters on the BG/Q, using up to 2,048 cores.

Notation. Let $\mathbf{A}, \mathbf{B}, \ldots$ denote matrices and let $\mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots$ denote column vectors. Given a vector $\mathbf{x} \in \mathbb{R}^m$, let $\|\mathbf{x}\|_2^2 = \sum_{i=1}^{m} \mathbf{x}_i^2$ be (the square of) its Euclidean norm; given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, let $\|\mathbf{A}\|_F^2 = \sum_{i,j=1}^{m,n} \mathbf{A}_{ij}^2$ be (the square of) its Frobenius norm. Let $\sigma_1 \geq \sigma_1 \geq \sigma_2 \cdots \geq \sigma_r > 0$ be the nonzero singular values of $\mathbf{A}$, where $r = rank(\mathbf{A})$ is the rank of the matrix $\mathbf{A}$. Then, the condition number of $\mathbf{A}$ is equal to $\kappa(\mathbf{A}) = \sigma_1/\sigma_r$.

II. The Blendenpik Algorithm for dense overdetermined systems

Blendenpik (see Algorithm 1) is a framework for least-squares solver for overdetermined, full column rank least-squares problems, which could be sparse or dense, that computes an approximate solution to the problem of eqn. (3), with a high degree of precision. Given a tall-and-thin matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, a column vector $\mathbf{b} \in \mathbb{R}^m$, and a regularizer λ, Blendenpik returns an approximate solution to eqn. (3) using the augmented input matrix $\mathbf{A}' = \begin{pmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{pmatrix}$ and the augmented vector $\mathbf{b}' = \begin{pmatrix} \mathbf{b} \\ 0 \end{pmatrix}$ by executing the following three steps:

1) A preconditioner is constructed by applying a randomized unitary (or approximately unitary) transform $\mathbf{F}$ to the augmented matrix $\mathbf{A}'$ and then sampling a small number of rows from the transformed matrix $\mathbf{FA}'$ to form the matrix $\mathbf{M}_s$.
2) A QR factorization of the sampled matrix $\mathbf{M}_s$ is computed, returning an orthogonal matrix $\mathbf{Q}_s$ and an upper triangular matrix $\mathbf{R}_s$. The latter matrix $\mathbf{R}_s$ is then used as a preconditioner for the augmented matrix $\mathbf{A}'$.
3) LSQR (an iterative method for solving least-squares problems) is then used to solve a least-squares problem using the preconditioned matrix to compute an approximate solution $\hat{\mathbf{x}}$ to the original problem of eqn. (3).

The algorithm uses a simple approach to estimate the condition number of the matrix $\mathbf{R}_s$. This procedure is described in [7] and amounts to computing the product $\|\mathbf{R}_s\|_1 \left\|\mathbf{R}_s^{-1}\right\|_1$, where $\|\mathbf{X}\|_1$ is the 1-norm of the matrix $\mathbf{X}$. If that estimate of the condition number is too small (and thus its inverse is too large), the algorithm tries to construct a new preconditioner. If no good preconditioner is constructed after three repetitions, the algorithm employs a standard solver to exactly compute the solution. We will use $\epsilon_{\text{machine}}$ to denote the target machine precision, which in our setting is equal to 2×10^{-15}.

Algorithm 1 The Blendenpik algorithm [7].

```
1:  Input: $\mathbf{A}' \in \mathbb{R}^{(m+n)\times n}$ matrix, $(m \gg n)$, $rank(\mathbf{A}') = n$, $\mathbf{b}' \in \mathbb{R}^{m+n}$, random transform matrix $\mathbf{F} \in \mathbb{R}^{(m+n)\times(m+n)}$, regularization parameter $\lambda > 0$, oversampling factor $\gamma \geq 1$ $(\gamma n \ll m+n)$.
2:  Output: approximate solution $\hat{\mathbf{x}}$ to the problem of eqn. (3).
3:  while TRUE do
4:      Let $\mathbf{S} \in \mathbb{R}^{(m+n)\times(m+n)}$ be a random diagonal matrix:
            $\mathbf{S}_{ii} = \begin{cases} 1, & \text{with probability } \frac{\gamma n}{m+n} \\ 0, & \text{otherwise} \end{cases}$
5:
6:      $\mathbf{M}_s = \mathbf{SFA}'$.
7:      Compute the QR factorization of $\mathbf{M}_s$: $\mathbf{M}_s = \mathbf{Q}_s\mathbf{R}_s$.
8:      Let $\hat{\kappa}$ be an estimate of the condition number of $\mathbf{R}_s$.
9:      if $\hat{\kappa}^{-1} > 5\epsilon_{\text{machine}}$ then
10:         $\mathbf{y} = \min_{\mathbf{z}} \left\|\mathbf{A}'\mathbf{R}_s^{-1}\mathbf{z} - \mathbf{b}'\right\|_2$.
11:         Solve $\mathbf{R}_s\hat{\mathbf{x}} = \mathbf{y}$ and return $\hat{\mathbf{x}}$.
12:     else if more than three iterations have been performed then
13:         Solve using the baseline least-squares solver and return.
14:     end if
15: end while
```

The most important stage of the Blendenpik algorithm is the application of the randomized transform $\mathbf{F}$; we will discuss various choices for $\mathbf{F}$ in Section II-A. It is worth emphasizing that computing $\mathbf{M}_s$ as the product $\mathbf{SFA}'$ in Algorithm 1 is for illustration purposes only. We will see in Section II-A that there are more efficient ways for computing $\mathbf{M}_s$ than simple matrix-matrix multiplication. The oversampling factor

[1]https://secure.cci.rpi.edu/wiki/index.php/Blue_Gene/Q

[2]The full source code of our batchwise Blendenpik implementation is available for download at https://github.com/cjiyer/libskylark/tree/batchwiseblendenpik.

γ guarantees that as the number of rows of $\mathbf{M}_s$ will be (in expectation and with high probability) close to γn. As a result, the computation of the QR decomposition of $\mathbf{M}_s$ is computationally efficient, since its running time only depends on the "small" dimension n and not on the "large" dimension m. The upper triangular matrix $\mathbf{R}_s$ that is returned by the QR decomposition of $\mathbf{M}_s$ is then used as a preconditioner for the original problem. However, if $\mathbf{R}_s$ is ill-conditioned, then we repeat the generation of the randomized transform $\mathbf{F}$ in the hope of getting a better-conditioned matrix $\mathbf{R}_s$. If this procedure fails three times, then an exact solver is employed to solve the regularized least-squares problem. We conclude this discussion by stating that while setting γ to a smaller value can improve the running time of the QR decomposition of $\mathbf{M}_s$, the quality of the preconditioner typically diminishes as γ decreases.

We now briefly discuss the LSQR method that is employed by Blendenpik in order to solve the preconditioned least-squares problem. LSQR [15] is an iterative, Krylov-subspace solver that works as follows: given the current iterate $\mathbf{x}_j$ and the corresponding residual error $\mathbf{r}_j = \mathbf{b}' - \mathbf{A}'\mathbf{x}_j$, LSQR uses the following criterion to test for convergence:

$$\frac{\left\| \left(\mathbf{A}'\mathbf{R}_s^{-1}\right)^T \mathbf{r}_j \right\|_2}{\left\|\mathbf{A}'\mathbf{R}_s^{-1}\right\|_F \left\|\mathbf{r}_j\right\|_2} \leq \rho,$$

where ρ is a tolerance value that determines the backward error at which the iterative solver terminates. This guarantees a backward stable solution to $\mathbf{y} = \min_{\mathbf{z}} \left\|\mathbf{A}'\mathbf{R}_s^{-1}\mathbf{z} - \mathbf{b}'\right\|_2$. The residual error at convergence is used to compute the final backward error estimate. The runtime of LSQR is affected by how well-conditioned the preconditioned system $\mathbf{A}'\mathbf{R}_s^{-1}$ is, which in turn is determined by the oversampling factor γ.

A. The Randomized Transform $\mathbf{F}$

Some of the earliest choices for the randomized transform $\mathbf{F}$ have been dense random matrices whose entries are independent random variables, chosen from various well-known distributions. For example, the simplest choice is to set the entries of $\mathbf{F}$ to be independent Gaussian random variables of zero mean and variance $1/(\gamma n)$. A second straight-forward choice is to set the entries of $\mathbf{F}$ to be $+1/\sqrt{\gamma n}$ or $-1/\sqrt{\gamma n}$ with probability 1/2, independently for each entry.

One of the most efficient and fastest-known dense random transform matrix $\mathbf{F}$ was proposed in the original Blendenpik paper [7]. The construction of $\mathbf{F}$ is the product of a random diagonal matrix and a fixed unitary transformation, namely the Discrete Cosine Transform (DCT). In this case, let $\mathbf{D}$ be a random diagonal matrix whose diagonal entries are set to +1 or -1 with probability 1/2. Then, let $\mathbf{C}$ be the matrix of the Discrete Cosine Transform (see [7] for details) and construct $\mathbf{F} = \mathbf{DC}$. We will refer to this construction of $\mathbf{F}$ as the Randomized DCT (RDCT). We note that, in this case, the computation of $\mathbf{FA}'$ is more efficient that matrix-matrix multiplication. Indeed, one can apply the DCT matrix $\mathbf{C}$ on the columns of $\mathbf{A}'$ in a column-wise manner much faster than matrix-vector multiplication, by using the properties of the Discrete Cosine Transform. Then, applying the matrices $\mathbf{S}$ and $\mathbf{D}$ on the resulting matrix $\mathbf{CA}'$ is trivial, since they are both diagonal matrices. Further, the DCT "smoothes out" localized information of $\mathbf{A}'$ after which applying $\mathbf{S}$ on the transformed matrix guarantees a sampled matrix with high probability.

However, dense randomized unitary transforms when applied to the sparse augmented matrix $\mathbf{A}'$ are quite inefficient since they do not take advantage of the matrix sparsity structure. A much better choice for the randomized transform $\mathbf{F}$ in the case of sparse matrices $\mathbf{A}'$ was proposed in the ground-breaking work of Clarkson and Woodruff [11]. We refer the reader to [11] for a detailed description of their construction; here we simply note that applying the resulting matrix $\mathbf{SF}$ on the augmented matrix $\mathbf{A}'$ takes time proportional to the sparsity of the augmented matrix $\mathbf{A}'$. We will refer to this construction of $\mathbf{F}$ as a Randomized Sparsity-Preserving Transform (RSPT).

Finally, we can set the randomized transform $\mathbf{F}$ to be a combination of the RSPT followed by a Randomized DCT (RDCT). Theoretically, such a combination leverages the sparsity-preserving properties of RSPT as well as the "smoothening" properties of the RDCT. However, as we shall observe in our evaluations, RSPT seems to construct a slightly better preconditioner as compared to the combination of RSPT and RDCT in the majority of our experiments.

III. Implementing our Algorithm on the Blue Gene/Q

Distribution formats for a 2-D process grid	$\sim$	$[M_C, M_R]$ & $[M_R, M_C]$ $[V_R, \star]$ & $[\star, V_R]$ $[V_C, \star]$ & $[\star, V_C]$ $[\star, \star]$
Distribution order within each grid dimension	M_C	Matrix column
	M_R	Matrix row
	V_C	Vector in column major order
	V_R	Vector in row major order
	$\star$	Stored on every process
Description	$[X, Y]$	Distribute [columns, rows] with scheme [X, Y]
	$[M_C, M_R]$	Distribute [columns, rows] equally among processes
	V_C/V_R	Distribute over processes in column/row major wrapping

TABLE I
Elemental data distribution overview.

The Blendenpik algorithm is implemented on top of the Elemental library [16]. Given a distributed environment over p processes, any dense matrix $\mathbf{A}' \in \mathbb{R}^{m\times n}$ is partitioned in Elemental into rectangular grids of sizes $r \times c$ in a 2D cyclic distribution, such that $p = r \times c$ and both r and c are $O(\sqrt{p})$. Elemental allows a matrix to be distributed in more than one way. An overview of various data distributions available in Elemental is given in Table I (not exhaustive). We use the standard distribution $[M_C, M_R]$ listed in Table I for dense matrices, in order to exploit operations that are communication intensive. For column-wise and row-wise vector operations that require local computations to be performed, we use a $[\star, V_C/V_R]$ or a $[V_C/V_R, \star]$ distribution that assigns each column or row vector to a single process. In some cases, we require a matrix or a column vector to be present across all processes, which is done using the $[\star, \star]$ format. The notations used henceforth are adapted from Elemental for convenience. [16] gives a comprehensive insight on these notations, describing different data distributions and the communication costs involved in redistribution.

The only construction of $\mathbf{F}$ that merits additional discussion is the Randomized Discrete Cosine Transform. In order to apply the RDCT on a matrix $\mathbf{A}'$ in a column-wise manner, we used the DCT implementation of FFTW [17], a highly optimized implementation of the Fast Fourier Transform (FFT), tuned for underlying architectures that work on multidimensional data. For our purposes, we used the 1-D versions of DCT that operate on Elemental's data distributions. In this case, the $[M_C, M_R]$ Elemental distribution is not a suitable format in order to apply FFTW's DCT, since the data distributed across multiple nodes in a column-wise as well as in a row-wise manner are locally non-contiguous. However, the implementation in FFTW expects contiguously distributed data across the relevant dimensions. We resolve this problem by redistributing the data so that all elements of a column or row vector are owned locally by a process, using either the $[V_R/V_C, *]$ or the $[*, V_R/V_C]$ distribution of Table I. In order to apply the DCT, all elements of a column must be stored locally, i.e., using the $[*, V_R/V_C]$ distribution. [18] gives a detailed overview on this redistribution procedure using a batchwise unitary transformation mechanism that is particularly suitable for dense matrices of terabyte sizes and above.

Elemental also supports a 1D distribution for sparse matrices where each process holds a fixed number of rows distributed roughly equally over p processes. We use this data distribution format for applying the Randomized Sparsity Preserving Transform (RSPT) of [11] with one caveat. Similar to the FFTW's DCT module, the RSPT is applied to columns of the sparse augmented matrix $\mathbf{A}'$ and hence all elements of a column must be present locally. We redistribute the sparse matrix using an `MPI_AlltoAll` call and then apply the RSPT to generate the sampled matrix $\mathbf{M}_s$.

The sampled matrix $\mathbf{M}_s$ generates the preconditioner $\mathbf{R}_s^{-1}$ obtained using Elemental's QR solver. Finally, the iterative solution is obtained using an LSQR implementation. The parameters for LSQR, the tolerance value ρ and the iteration limit N_{iter} can be suitably tuned depending upon the magnitude of accuracy needed and the speedup desired from our Blendenpik solver. Usually a tolerance value ρ closer to $\epsilon_{\text{machine}}$ is chosen for a better backward stable solution.

IV. Evaluation

We relied on the UFL Sparse matrix collection [19] to obtain matrices of different condition numbers for our evaluations. We chose to work with the following matrices: **(i)** the ns3Da matrix[3]; **(ii)** the mesh deform matrix[4]; **(iii)** the memplus matrix[5]; **(iv)** the sls matrix[6]; **(v)** the rma10 matrix[7]; and **(vi)** the c-41 matrix[8].

We replicated and vertically concatenated each of the sparse matrices (with different random noise added at each replication step) in order to create tall-and-thin sparse matrices $\mathbf{A}_{rep}$. Table II shows the specific matrices that were used in our evaluations: starting with the aforementioned matrices, we slightly densified the replicated matrices by adding a random Gaussian noise matrix $\mathbf{E} \sim \mathcal{N}(0,1)$ to the replicated matrix $\mathbf{A}_{rep}$. The number of non-zero entries in the slightly densified replicated matrix $\mathbf{A}_{rep} + \mathbf{E}$ is equal to the number of non-zero entries in $\mathbf{A}_{rep}$ plus the number of non-zero entries in $\mathbf{E}$, which is upper bounded by the product of the dimensions of $\mathbf{E}$ divided by 1,000.

Each matrix in Table II is suffixed using the number of replicates (essentially the number of vertical concatenations that we applied to the "base" sparse matrix in order to created a tall-and-thin matrix). For example, to construct the matrix ns3Da-8, we started with the ns3Da matrix and created eight copies; then, we vertically concatenated these eight copies to get a tall-and-thin matrix; finally, we added the random normal noise matrix $\mathbf{E}$ as described above.

Matrix Name	**# of rows**	**# of columns**	**# of entries (Millions)**	**Condition number**
ns3Da-8	$163,312$	$20,414$	16.77	$7.07E+002$
mesh deform-4	$936,092$	$9,393$	12.2	$1.17E+003$
memplus-32	$568,256$	$17,758$	13.26	$1.29E+005$
sls-1	$1,748,122$	$62,729$	116.462	$8.67E+007$
rma10-8	$374,680$	$46,835$	36.18	$7.98E+010$
c-41-32	$312,608$	$9,769$	6.3	$4.78E+012$

TABLE II
Matrices used in our evaluations.

We now describe our evaluation metrics. Recall that $\mathbf{A}' \in \mathbb{R}^{(m+n)\times n}$ is the augmented matrix and $\mathbf{b}' \in \mathbb{R}^{m+n}$ is the augmented target vector. Let $\hat{\mathbf{x}}$ be the approximate solution returned by Algorithm 1 and let $\mathbf{x}^*$ be the optimal solution to the problem of eqn. (3). Let $\hat{t}_{\text{run}}$ be the running time of Algorithm 1 and let t^*_{run} be the running time of the baseline Elemental sparse least-squares solver. Then, our first accuracy metric is the relative error of the approximate solution $\hat{\mathbf{x}}$, given by the following formula:

$$\left\|\mathbf{A}'\hat{\mathbf{x}} - \mathbf{A}'\mathbf{x}^*\right\|_2 / \left\|\mathbf{A}'\mathbf{x}^*\right\|_2. \tag{4}$$

We also compute the backward error of the approximate solution as follows:

$$\left\|\mathbf{A}'^T\left(\mathbf{b}' - \mathbf{A}'\hat{\mathbf{x}}\right)\right\|_2. \tag{5}$$

Finally, the speedup of Algorithm 1 is defined as

$$t^*_{\text{run}}/\hat{t}_{\text{run}}. \tag{6}$$

We tune AMOS to evaluate the Blendenpik implementation against optimum baseline performance. The choice of 1 MPI process and 32 OpenMP threads per Blue Gene/Q node was the standard configuration that we selected for our evaluations that gave maximum performance for the baseline Elemental sparse solver on AMOS. We used the `bgclang/LLVM` runtime environment for our evaluations.

[3]http://www.cise.ufl.edu/research/sparse/matrices/FEMLAB/ns3Da.html
[4]http://www.cise.ufl.edu/research/sparse/matrices/Yoshiyasu/mesh_deform.html
[5]http://www.cise.ufl.edu/research/sparse/matrices/Hamm/memplus.html
[6]http://www.cise.ufl.edu/research/sparse/matrices/Bates/sls.html
[7]https://www.cise.ufl.edu/research/sparse/matrices/Bova/rma10.html
[8]http://www.cise.ufl.edu/research/sparse/matrices/Schenk_IBMNA/c-41.html

A. Baseline

The baseline Elemental sparse solver solves the regularized least squares problem for overdetermined matrices given by eqn. (3) by applying a priori regularization to symmetric quasi-semidefinite augmented systems. The augmented systems are of special interest since they reduce to quasi-definite matrices which can then be solved by a Cholesky-based factorization approach, like the LDL decomposition [20]. For ill-conditioned matrices, the baseline solver provides two tuning parameters:

1) The a priori regularizer α. An α value close to σ_r, where $r = \text{rank}(\mathbf{A})$ is recommended. We typically set $\alpha = \epsilon_{\text{machine}}$.
2) A temporary regularizer γ_0 set to $\epsilon_{\text{machine}}^{1/4}$.

The work of [16] gives comprehensive details on the aforementioned reduction to quasi-definite forms and the distributed Cholesky-based sparse solvers.

B. The effect of the regularization λ

Our objective in this paper is to provide a thorough evaluation of Blendenpik-type solvers for sparse ridge-regression problems. We specifically seek to analyze the impact of the regularization parameter λ from three primary viewpoints: scalability, performance, and accuracy, with respect to the evaluation metrics given in eqns. (4), (5) and (6). We aim to understand the performance of the sparse randomized transforms $\mathbf{F}$ described in Section II-A on Blendenpik-type solvers for the problem of eqn. 3. We also seek to understand the impact of our choice for $\mathbf{F}$ in terms of strong scaling on Blendenpik for the optimal regularizer. Finally, we will evaluate the impact of the oversampling factor γ on the Blendenpik algorithm using the aforementioned metrics. At this point, we set γ to four for evaluating our randomized transforms and we will validate this choice for γ later in Section IV-D. All our evaluations were run on 128 BG/Q nodes on AMOS.

1) Scalability Evaluation: To demonstrate that the Blendenpik algorithm is a scalable solver for sparse overdetermined least-squares systems, we analyze the speedup of eqn. (6) for the RSPT, and for a combination of the RSPT followed by the RDCT (henceforth referred to as RSPT-RDCT), over the baseline Elemental solver for various choices of the regularization parameter λ. Figures 1 and 2 show the impact of regularization on both well-conditioned and ill-conditioned matrices for both sparse transforms. *It is worth noting that the baseline solver fails to converge for the largest matrix* `sls-1`.

As seen in figure 1, both RSPT and RSPT-RDCT show excellent speedups for the well-conditioned matrices `ns3Da-8`, `mesh_deform-4`, and `memplus-32`. Further, as the regularization parameter λ increases, the speedup increases until it peaks for a certain choice of the regularization parameter. Then it starts dropping, indicating the existence of a point of diminishing returns. However, the value of the regularization parameter where maximum speedups occur is different for our various input matrices and seems to correlate with their condition numbers. Indeed, the regularization value where maximum speedup is achieved increases as the condition numbers of the input matrices increase. This is because as the matrix gets more ill-conditioned, the preconditioner constructed is much less well-conditioned, and hence the time spent by the LSQR solver to converge also increases. As the regularization value increases for the ill-conditioned matrix, the preconditioner constructed is much better-conditioned leading to a faster convergence of the LSQR stage.

Another key observation is that the speedup achieved by RSPT is always better than the speedup achieved by the RSPT-RDCT transform. There are two reasons underlying the slower speedup of the RSPT-RDCT transform. First, the Blendenpik algorithm spends a reasonable amount of time to compute the RDCT transform. Second, the RSPT always produces a marginally better preconditioner than the RSPT-RDCT transform, thus leading to faster convergence at the LSQR stage. We emphasize here that the sketch size of the RDCT is the same as that of the RSPT.

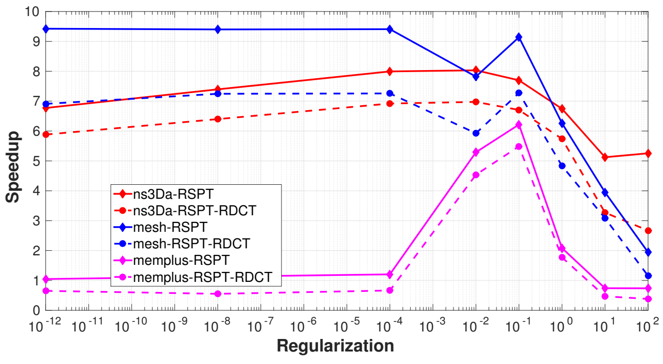

Fig. 1. Speedup of Blendenpik solver over the parallel baseline sparse solver for well-conditioned sparse matrices as a function of increasing regularization values.

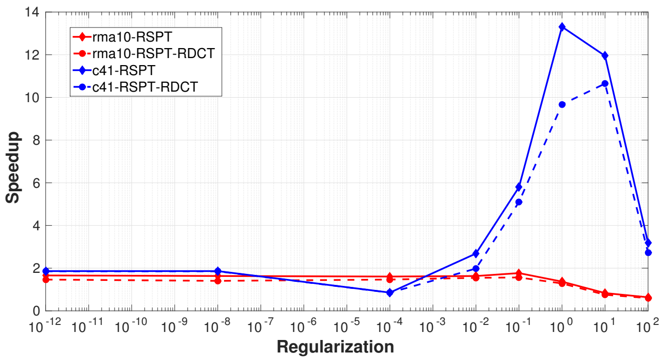

Fig. 2. Speedup of Blendenpik solver over the parallel baseline sparse solver for ill-conditioned sparse matrices as a function of increasing regularization values.

Figure 2 shows the speedup of the sparse transforms for the ill-conditioned matrices `rma10-8` and `c-41-32`. We observe that the speedup for the `rma10-8` matrix is quite poor, even as the regularization parameter increases. This happens because the LSQR solver stagnates as the number of iterations increases. However, for the more highly ill-conditioned matrix `c-41-32`, the behavior is quite different as the speedup increases with increasing values of the regularization parameter. Again, the speedup peaks at a certain point and then drops off again. The peak speedup obtained for the c−41−32 matrix is much higher than the rma10 matrix, which indicates that the residual vectors in the iterations of LSQR eventually become linearly dependent.

2) Numerical Stability Evaluation: We evaluate the numerical stability of the Blendenpik solver for RSPT and RSPT-RDCT for increasing values of the regularization parameter λ. The numerical stability is captured by the relative error (see eqn. (4)) and the backward error (see eqn. (5)). Figures 3 and 4 show the effect of increasing values of the regularization parameter on the relative error for well-conditioned and ill-conditioned matrices respectively. In both cases, the relative error for the sparse randomized transforms decreases as regularization increases. For the well-conditioned matrices in figure 3, the optimal value of the relative error occurs at $\lambda = 0.1$ for both transforms. For the ill-conditioned matrices, the optimal regularization value that results in smallest relative error is greater than that for the well-conditioned ones, which for rma10−8 is $\lambda = 10^2$ and for c−41−32 is $\lambda = 10$. These are the optimal regularization values that we choose for our various datasets to measure performance for our strong scaling and oversampling evaluations henceforth denoted by $\boldsymbol{\lambda^*}$.

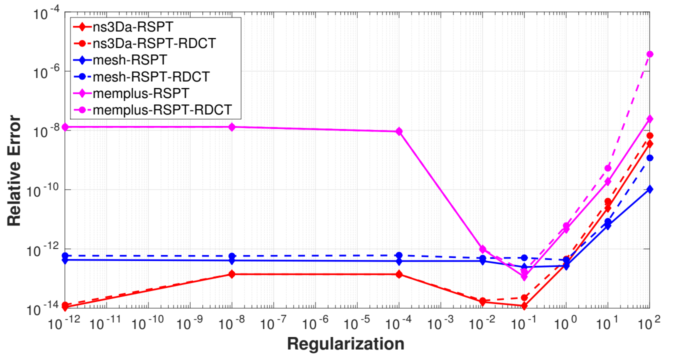

Fig. 3. Relative Error for well-conditioned sparse matrices as a function of λ.

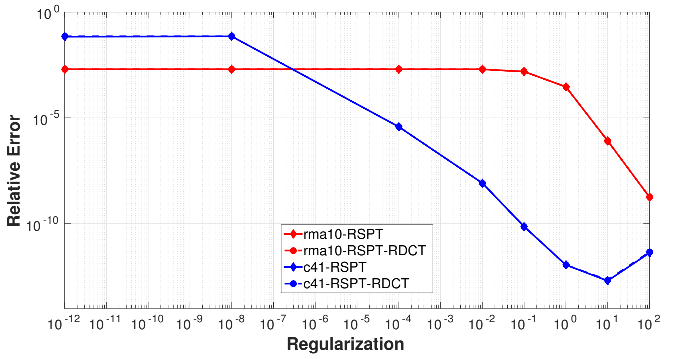

Fig. 4. Relative Error for ill-conditioned sparse matrices as a function of λ.

Finally, we show the effect of regularization on backward error for well-conditioned and ill-conditioned matrices in figures 5 and 6. The backward error decreases marginally as regularization increases for both well-conditioned and ill-conditioned matrices. The randomized transforms have comparable backward error when compared to the baseline sparse solver for all matrices. Another interesting observation is that the backward error achieved by the sparse randomized transforms for both well-conditioned and ill-conditioned matrices differs approximately by condition number order of magnitude difference.

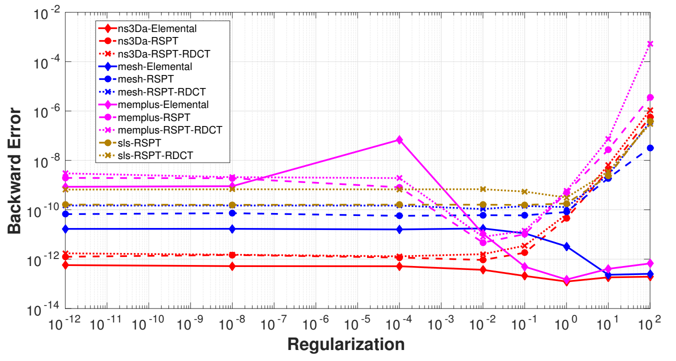

Fig. 5. Backward Error for well-conditioned sparse matrices as a function of λ.

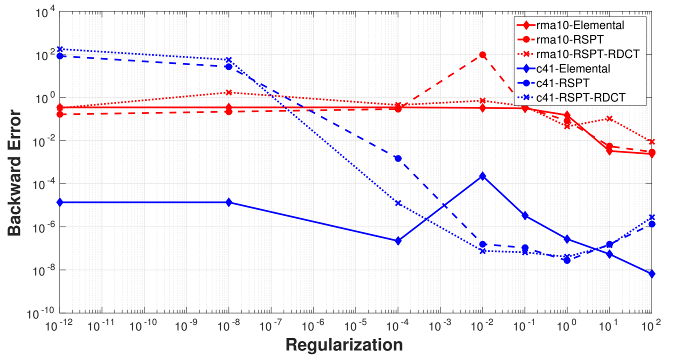

Fig. 6. Backward Error for ill-conditioned sparse matrices as a function of λ.

C. *Performance evaluation at λ^**

We also evaluate the strong scaling performance of the Blendenpik algorithm as a function of the (increasing) number of the Blue Gene/Q nodes. Figure 7 shows the strong scaling performance of the sparse randomized sketching transforms for all matrices. We observe that with the exception of rma10 matrix, the sparse randomized solvers show almost linear strong scaling with increasing BG/Q nodes for the optimal regularization value λ^*. Further, RSPT outperforms RSPT-RDCT for all matrices for increasing BG/Q nodes, which again shows that RSPT constructs a better preconditioner than RSPT-RDCT for all matrices irrespective of their condition numbers. The bottleneck for the Blendenpik algorithm at λ^* is the QR stage which constructs the preconditioner. As the number of BG/Q nodes increase, QR scales proportionally for a fixed problem size, while the reduction to quasi-definite form in the sparse least-squares solver is quite inefficient and does not scale with as the number of nodes increases.

D. *The effect of the oversampling factor γ at λ^**

An important choice in the construction of an efficient preconditioner in the context of the Blendenpik algorithm is

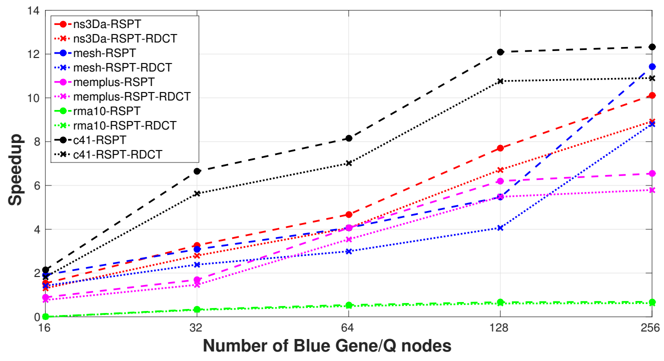

Fig. 7. Strong Scaling as a function of increasing Blue Gene/Q nodes at optimal regularization value λ^*.

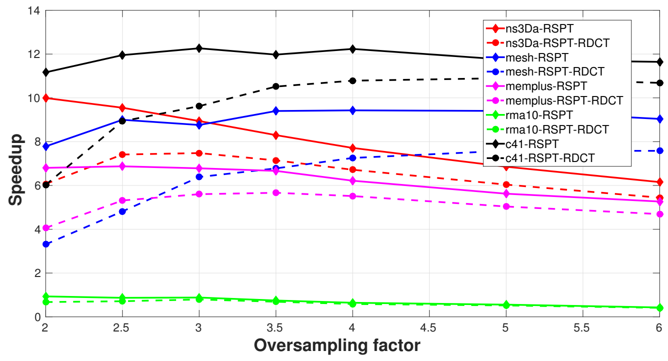

Fig. 8. Speedup of Blendenpik solver over the parallel baseline sparse solver as a function of increasing oversampling factors at optimal regularization value λ^*.

the value of the oversampling factor γ that decides the number of rows (equal, in expectation, to γn) of the preconditioner. Of particular interest is an analysis of the behavior of the sparse randomized transforms in the Blendenpik solver with respect to the metrics described in Section IV as a function of γ. We evaluate the Blendenpik solver on the various well-conditioned and ill-conditioned matrices on 128 BG/Q nodes as a function of γ, where γ ranges between 2.0 and 6.0 in increments of 0.5. We seek to understand the effect of γ on the scalability and the numerical stability of the Blendenpik algorithm at the optimal regularization value λ^* for each matrix. As mentioned earlier, the baseline sparse Elemental solver fails to converge for the largest matrix `sls−1` and hence we exclude its analysis from this section.

Figure 8 shows the speedup of the Blendenpik algorithm for increasing values of the oversampling factor γ for the sparse sketching transforms. Figure 8 reveals several interesting observations as the oversampling factor γ increases. The speedup of the RSPT sketching transform marginally increases for increasing values of γ for all matrices. However, for the RSPT-RDCT transform, the speedup rapidly increases when γ increases which suggests that the oversampling factor γ plays a much more significant role in the speedup of the RDCT stage and the faster convergence of the RSPT-RDCT preconditioner. However, for all values of γ, RSPT marginally outperforms the RSPT-RDCT combination for all matrices, primarily due to the time spent in the RDCT batchwise transform stage. The `rma10−8` matrix however shows sublinear performance as compared to the other matrices at the optimal regularization value λ^*, mainly due to the stagnation of the LSQR algorithm mentioned earlier for the sparse sketching transforms.

As discussed in Section IV-B2, the numerical stability is measured in terms of relative and backward error. Figure 9 shows the relative error for both sparse transforms as the oversampling factor γ increases. The relative error marginally decreases with increasing values of γ for all matrices. The Blendenpik solver is fairly stable with respect to the relative error at λ^*, which is close to 12 to 14 digits of accuracy for all matrices, with the exception of the `rma10−8` matrix which exhibits around nine digits of accuracy. The RSPT again exhibits better relative error stability than the RSPT-RDCT combination for all matrices for increasing values of γ.

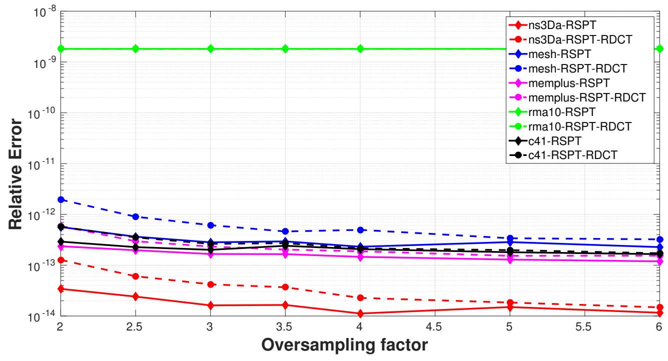

Fig. 9. Relative Error as a function of increasing oversampling factors at optimal regularization value λ^*.

Finally figure 10 shows the behavior of backward error as a function of the oversampling factor γ at λ^*. The well-conditioned matrices `ns3Da−8`, `mesh_deform−4`, `memplus−32` and `sls−1` exhibit significantly better backward error stability than the ill-conditioned matrices `c−41−32` and especially `rma10−8`. That being said, the backward error for the sparse randomized transforms are still comparable (within an order of magnitude) to the backward error of the baseline sparse solver for all matrices. Further, the backward error for both sparse transforms decreases marginally with increasing γ values for all matrices.

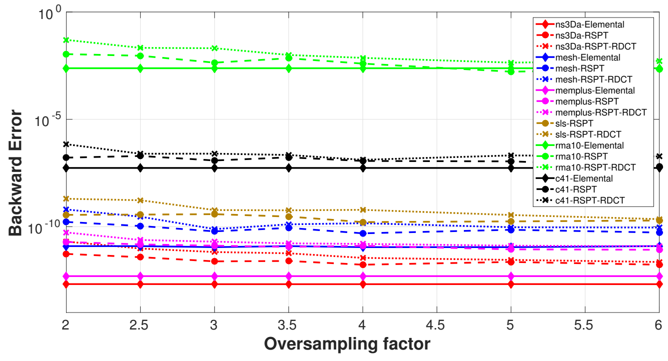

Fig. 10. Backward Error as a function of increasing oversampling factors at optimal regularization value λ^*.

E. Summarizing our empirical evaluations

To help the reader parse our extensive empirical evaluations, we briefly summarize our findings. **(i)** The speedup achieved by RSPT is always better than the RSPT-RDCT transform. There are two reasons for this slower speedup for the RSPT-RDCT sparse transform. First, the Blendenpik algorithm spends reasonable time to compute the RDCT transform. Second, the RSPT always produces a marginally better preconditioner than the RSPT-RDCT transform that leads to faster convergence of the LSQR stage. **(ii)** As the regularization values increase, the speedup increases until it peaks for a certain regularization value and then reduces again for all matrices with the exception of the `rma10-8` matrix. **(iii)** The relative error decreases with increasing values of the regularization parameter until it achieves the smallest relative error at $\lambda = \lambda^*$. We choose this regularization value λ^* as the optimal regularizer for our strong scaling and oversampling evaluations. As the condition numbers of the matrices increase, λ^* for each matrix also increases. **(iv)** The sparse randomized transforms have comparable backward error compared against the baseline sparse solver for all matrices. The backward error achieved by the Blendenpik solver for different matrices differ approximately by the same order of magnitude as the difference in the condition numbers of the matrices. **(v)** The sparse randomized transforms demonstrate significant strong scaling for all matrices at λ^* with the exception of the `rma10-8` matrix. **(vi)** The Blendenpik solver demonstrates excellent speedup and numerical stability in terms of the relative error at λ^* for increasing oversampling factors. The backward error is somewhat worse yet comparable to the backward error achieved by the baseline sparse Elemental solver at λ^*.

V. Conclusions and Future work

In this paper we proposed and demonstrated a highly scalable distributed memory least squares solver based on the Blendenpik algorithm using sparse randomized transforms. Our solver outperforms a state-of-the-art parallel sparse solver both in runtime and in the dimensions of the matrices that can be solved. In the future, we plan to develop similar scalable algorithms using sparse transforms for variants of ridge regression like dual ridge regression and kernel ridge regression.

Acknowledgments

We would like to thank Jack Poulson for numerous helpful discussions and his constant and timely support with the Elemental library.

References

[1] A. Bjőrck, *Numerical Methods for Least Squares Problems*. Siam Philadelphia, 1996.

[2] C. Saunders, A. Gammerman, and V. Vovk, "Ridge regression learning algorithm in dual variables," in *Proceedings of the Fifteenth International Conference on Machine Learning*, ser. ICML '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 515–521. [Online]. Available: http://dl.acm.org/citation.cfm?id=645527.657464

[3] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[4] T. Sarlos, "Improved Approximation Algorithms for Large Matrices via Random Projections," in *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 143–152. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2006.37

[5] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós, "Faster least squares approximation," *Numer. Math.*, vol. 117, no. 2, pp. 219–249, Feb. 2011. [Online]. Available: http://dx.doi.org/10.1007/s00211-010-0331-6

[6] V. Rokhlin and M. Tygert, "A fast randomized algorithm for overdetermined linear least-squares regression," *Proc. Natl. Acad. Sci. USA*, vol. 105, no. 36, pp. 13 212–13 217, 2008.

[7] H. Avron, P. Maymounkov, and S. Toledo, "Blendenpik: Supercharging LAPACK's Least-Squares Solver." *SIAM J. Scientific Computing*, vol. 32, no. 3, pp. 1217–1236, 2010. [Online]. Available: http://dblp.uni-trier.de/db/journals/siamsc/siamsc32.html#AvronMT10

[8] J. Yang, X. Meng, and M. W. Mahoney, "Implementing Randomized Matrix Algorithms in Parallel and Distributed Environments," *CoRR*, vol. abs/1502.03032, 2015. [Online]. Available: http://arxiv.org/abs/1502.03032

[9] Y. Lu, P. Dhillon, D. P. Foster, and L. Ungar, "Faster ridge regression via the subsampled randomized hadamard transform," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 369–377.

[10] S. Chen, Y. Liu, M. R. Lyu, I. King, and S. Zhang, "Fast relative-error approximation algorithm for ridge regression," in *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, 2015, pp. 201–210.

[11] K. L. Clarkson and D. P. Woodruff, "Low Rank Approximation and Regression in Input Sparsity Time," in *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '13. New York, NY, USA: ACM, 2013, pp. 81–90. [Online]. Available: http://doi.acm.org/10.1145/2488608.2488620

[12] X. Meng, M. A. Saunders, and M. W. Mahoney, "LSRN: A parallel iterative solver for strongly over- or under-determined systems," *CoRR*, vol. abs/1109.5981, 2011. [Online]. Available: http://arxiv.org/abs/1109.5981

[13] M. P. Forum, "MPI: A message-passing interface standard," Knoxville, TN, USA, Tech. Rep., 1994.

[14] L. Dagum and R. Menon, "OpenMP: An Industry-Standard API for Shared-Memory Programming," *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Jan. 1998. [Online]. Available: http://dx.doi.org/10.1109/99.660313

[15] C. C. Paige and M. A. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Trans. Math. Softw.*, vol. 8, no. 1, pp. 43–71, Mar. 1982. [Online]. Available: http://doi.acm.org/10.1145/355984.355989

[16] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, "Elemental: A New Framework for Distributed Memory Dense Matrix Computations," *ACM Trans. Math. Softw.*, vol. 39, no. 2, pp. 13:1–13:24, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2427023.2427030

[17] M. Frigo and S. G. Johnson, "FFTW: An adaptive software architecture for the FFT," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, Seattle, Washington, 1998, pp. 1381–1384.

[18] C. Iyer, H. Avron, G. Kollias, Y. Ineichen, C. Carothers, and P. Drineas, "A scalable randomized least squares solver for dense overdetermined systems," in *Proceedings of the 6th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ser. ScalA '15. New York, NY, USA: ACM, 2015, pp. 3:1–3:8. [Online]. Available: http://doi.acm.org/10.1145/2832080.2832083

[19] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, dec 2011. [Online]. Available: http://doi.acm.org/10.1145/2049662.2049663

[20] M. A. Saunders, "Cholesky-based Methods for Sparse Least Squares: The Benefits of Regularization," *Linear and Nonlinear Conjugate Gradient-Related Methods*, pp. 92–100, Feb. 1996. [Online]. Available: http://www.stanford.edu/group/SOL/papers/seattleproc.pdf